

版权相关注意事项：

- 1、书籍版权归著者和出版社所有
- 2、本PDF来自于各个广泛的信息平台，经过整理而成
- 3、本PDF仅限用于非商业用途或者个人交流研究学习使用
- 4、本PDF获得者不得在互联网上以任何目的进行传播
- 5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
- 6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
- 7、请于下载PDF后24小时内研究使用并删掉本PDF

· 开发宝典丛书 ·

以Xcode 6为开发环境，详细讲解全新的iOS 8应用开发
通过121个实例全面展现iOS开发中较为深入的12类应用模块

iOS 开发 范例实战宝典

(进阶篇)

杨佩璐 魏彩娟 刘媛媛 编著



(附赠51CTO学院学习卡)

- ✓ **实例丰富：**详细讲解121个iOS经典实例的开发过程，提高实战开发水平
- ✓ **涵盖广泛：**涵盖图形图像、图表、动画、网页、地图、音频/视频、触摸、手势、传感器和网络等12类应用模块
- ✓ **由浅入深：**实例按照“实现原理、实现过程、重点代码”的编排顺序讲解，很容易掌握
- ✓ **代码精简：**精简结构性代码，只保留关键代码和核心代码，以节省篇幅，让本书更超值
- ✓ **配流程图：**为复杂的实例配有详细的程序流程图，以帮助读者轻松理解程序的执行过程
- ✓ **重点讲解：**对每个实例的核心功能都给予了专门讲解，以便于读者更好地掌握
- ✓ **最新技术：**书中的实例完全适用于全新的iOS 8开发平台，也兼容iOS 7开发平台
- ✓ **答疑解惑：**提供了QQ群、技术论坛和E-mail等完善的学习交流和沟通方式（见前言中的说明）

清华大学出版社



作者简介

杨佩璐 现任教于山东中医药大学理工学院，副教授。研究方向为计算机科学技术、计算机应用和移动设备应用开发。长期从事计算机相关课程的教学和课题研究，发表了多篇计算机方面的论文，并出版了多部计算机类图书。

魏彩娟 毕业于解放军信息工程大学网络工程专业。现就职于河南牧业经济学院，从事计算机相关课程的教学工作。对iOS和Android移动开发技术有浓厚的兴趣，并有深入的研究。

刘媛媛 软件工程师。现就职于某知名IT公司的移动项目部，担任iOS开发技术研究员。擅长iOS移动开发技术，长期从事iOS新技术和可穿戴设备的研究。参与过多个移动项目的开发，积累了丰富的开发经验。编写并出版了《Swift入门很简单》、《Xamarin iOS移动开发实战》和《Swift游戏开发案例实战》等图书。

· 开发宝典丛书 ·

iOS 开发 范例实战宝典

(进阶篇)

杨佩璐 魏彩娟 刘媛媛 编著

清华大学出版社
北 京

书籍是人类进步的阶梯

内 容 简 介

《iOS 开发范例实战宝典》分为基础篇和进阶篇两个分册。其内容包含了 iOS 开发必知必会的 238 个经典实例和几百个开发模块。书中的实例紧跟技术趋势，以最新的 iOS 8 为版本编写，内容覆盖了 iOS 开发的方方面面，几乎涉及 iOS 开发的所有重要知识。书中给出了每个实例的具体实现过程，并对程序代码做了详细注释，对其中的重点和难点进行了专门分析，而且精讲每个实例的重点代码，读者可以在这些实例的基础上做出更多更新的功能。

本书为《iOS 开发范例实战宝典（进阶篇）》，共 12 章，包含了 121 个开发实例。其中包括 59 个图形图像类实例、5 个图标类实例、16 个动画类实例、9 个网页视图类实例、13 个地图类实例、9 个音频和视频类实例、4 个内部应用程序类实例、8 个触摸和手势类实例、10 个照片库与相机类实例、4 个传感器类实例和 4 个网络类实例。

本书涉及面广，涉及 iOS 软件开发的各种常用应用。适合所有想全面学习 iOS 开发技术的人员阅读，也适合 iOS 专业开发人员作为案头必备的参考书。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目（CIP）数据

iOS 开发范例实战宝典（进阶篇）/ 杨佩璐，魏彩娟，刘媛媛编著. —北京：清华大学出版社，2015
（开发宝典丛书）

ISBN 978-7-302-39702-1

I. ①i… II. ①杨… ②魏… ③刘… III. ①移动终端—应用程序—程序设计 IV. ①TN929.53

中国版本图书馆 CIP 数据核字（2015）第 061849 号

责任编辑：杨如林

封面设计：欧振旭

责任校对：徐俊伟

责任印制：何 芊

出版发行：清华大学出版社

网 址：http://www.tup.com.cn, http://www.wqbook.com

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社总机：010-62770175 邮 购：010-62786544

投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn

质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

印刷者：清华大学印刷厂

装订者：三河市新茂装订有限公司

经 销：全国新华书店

开 本：185mm×260mm

印 张：36.75

字 数：918 千字

版 次：2015 年 5 月第 1 版

印 次：2015 年 5 月第 1 次印刷

印 数：1~3000

定 价：99.00 元

产品编号：063391-01

前 言

移动应用开发是现在 IT 开发的热点。由于苹果提供了完备的开发工具和成熟的软件盈利方式，苹果的 iOS 开发成为热点中的热点。苹果开发技术较为封闭，尤其是相对于开源技术 Android 开发。同时由于移动开发发展时间较短，导致开发资料相对匮乏，开发者往往缺乏应用指导性资料。

笔者结合自己多年的 iOS 开发经验和心得体会，花费了一年多的时间分析了 iOS 开发中常见的几百个应用场景，并进行了精心整理，挑选了最为典型的 238 个 iOS 开发实例，编写成了《iOS 开发范例实战宝典》（分为基础篇和进阶篇两个分册）。

本书为《iOS 开发范例实战宝典（进阶篇）》，包含了 121 个经典实例，涉及 iOS 开发中较为深入的 12 个应用开发专题。希望各位读者能在本书的引领下跨入 iOS 开发的大门，并成为一名开发高手。

本书特色

1. 实例丰富，代码精讲

本书详细讲解了 121 个 iOS 开发经典实例，并对重点代码做了大量注释和讲解，以便于读者更加轻松地学习。通过对这些实例的演练，读者可以快速提高开发水平。

2. 内容全面，涵盖广泛

本书全面介绍了 iOS 开发中最为常见的 12 类应用开发模块，包括图形、图像、图表、动画、网页视图、地图、音频、视频、内置应用程序、触摸、手势、照片库、相机、传感器、网络。这些内容是 iOS 开发必知必会的内容，需要读者重点掌握。

2. 讲解详细，循序渐进

本书中的每个实例都给出了详细的分析过程和实现步骤，书中的每个实例都按照“实现原理→实现过程→重点代码”3 个步骤进行分析。对于复杂的实例，还给出了完备的流程图帮助读者理解实例的工作机制，掌握起来更加容易。

4. 专注核心，举一反三

为了在有限的篇幅内讲解更多的开发实例，在本书中只给出了每个实例的核心代码及分析。完整的实例代码读者可以自己阅读，并进行测试和练习，而且还可以对这些代码进行改造，以用于实际的开发之中，从而起到举一反三的作用。

本书内容及体系结构

第1章 图形图像（一）

本章 20 个实例，主要内容包括：图片浏览器、疯狂成语、猜老猫、图片编辑器、湖中倒影、颈部运动、翻翻看等内容。通过本章的学习，读者可以掌握关于图形图像的一些常见技术与应用。

第2章 图形图像（二）

本章 19 个实例，主要内容包括：简易相框、图像滤镜、图像的点击放大、万花筒、浏览商品图片、具有放大镜的图像、照片墙等内容。通过本章的学习，读者可以进一步掌握关于图形图像的一些常见技术与应用。

第3章 图表

本章 5 个实例，主要内容包括：饼状图、柱状图、折线图、波形图、油量表。通过本章的学习，读者可以掌握关于图表的一些常见技术与应用。

第4章 动画

本章 16 个实例，主要内容包括：飘落的雪花、自动旋转的太极、礼花效果、物理引擎——掉落的蘑菇、物理引擎——橡皮筋、吃豆豆、打砖块、碰撞的火球等内容。通过本章的学习，读者可以掌握关于动画的一些常见技术与应用。

第5章 网页视图

本章 9 个实例，主要内容包括：紧急求救中心、常用网址大全、改变网页视图中字体的大小、网页视图的背景透明化、网页的下拉刷新、天气预报、城市地理信息查询、滑动网页时，隐藏工具栏、网页浏览器。通过本章的学习，读者可以掌握关于网页视图的一些常见技术与应用。

第6章 地图

本章 13 个实例，主要内容包括：地图切换器、温带换算器、地图导航、位置跟踪器、指南针、驴友历程、地图的位置查找、3D 地图、旋转的地图等内容。通过本章的学习，读者可以掌握关于地图的一些常见技术与应用。

第7章 音频和视频

本章 9 个实例，主要内容包括：小钢琴、手机铃声变化器、十种语言、播放歌曲的同时显示歌词、录音机、获取系统中所有的音频文件、讯飞识别、音乐播放器、视频播放器。通过本章的学习，读者可以掌握关于音频和视频的一些常见技术与应用。

第 8 章 内置的应用程序

本章 4 个实例，主要内容包括：工作日计算器、短信发送、日历、添加录。通过本章的学习，读者可以掌握关于 iOS 内置的应用程序的一些常见技术与应用。

第 9 章 触摸和手势

本章 8 个实例，主要内容包括：打地鼠、人鱼公主换发记、被挤扁的气球、撕裂图像、一个手指实现缩放、仿小米手机的解锁功能、QQ 的解锁功能、拖动选择图片墙。通过本章的学习，读者可以掌握关于触摸和手势的一些常见技术与应用。

第 10 章 照片库与相机

本章 10 个实例，主要内容包括：更改应用程序的背景、自定义相机、狙击枪、水印相机、QQ 聊天视频效果、iOS 7 手电筒实现、三连拍等内容。通过本章的学习，读者可以掌握关于照片库和相机的一些常见技术与应用。

第 11 章 传感器

本章 4 个实例，主要内容包括：手机水平放置的测试、加速的小球、摇一摇音乐播放器、根据手机转动显示图像。通过本章的学习，读者可以掌握关于传感器的一些常见技术与应用。

第 12 章 网络

本章 4 个实例，主要内容包括：手机号码查询、在 Safari 中打开 URL、后台下载测试、图像下载队列控制器。通过本章的学习，读者可以掌握关于网络的一些常见技术与应用。

本书读者对象

- ☐ 想全面学习 iOS 开发技术的人员；
- ☐ iOS 专业开发人员；
- ☐ iOS 开发爱好者；
- ☐ 大中专院校的学生；
- ☐ 社会培训班学员；
- ☐ 需要一本案头必备手册的程序员。

本书配套资源获取方式

本书提供以下的配套资源：

- ☐ 本书开发环境；
- ☐ 本书实例源代码。

为了节省读者的购书开支，本书放弃以配书光盘的方式提供这些资源，而是采用提供

下载的方式。读者可以登录清华大学出版社网站（www.tup.com.cn），搜索到本书页面，然后按照提示下载，也可以在本书服务网站（www.wanjuanchina.net）的相关版块上下载这些配套资源。

本书售后服务方式

编程学习的最佳方式是共同学习。但是由于实际环境所限，大部分读者都是独自前行。为了便于读者更好地学习 iOS 语言，我们构建了多样的学习环境，力图打造立体化的学习方式，除了对内容精雕细琢之外，还提供了完善的学习交流和沟通方式。主要有以下几种方式：

- ❑ 提供技术论坛 <http://www.wanjuanchina.net>，读者可以将学习过程中遇到的问题发布到论坛上以获得帮助。
- ❑ 提供 QQ 交流群 336212690，读者申请加入该群后便可以和作者及广大读者交流学习心得，解决学习中遇到的各种问题。
- ❑ 提供 book@wanjuanchina.net 和 bookservice2008@163.com 服务邮箱，读者可以将自己的疑问发电子邮件以获取帮助。

本书作者

本书主要由山东中医药大学的杨佩璐、河南牧业经济学院的魏彩娟和大学霸网站的刘媛媛编写。其中，杨佩璐编写了本书的第 1~5 章，魏彩娟编写了本书的第 6~9 章，刘媛媛编写了本书的第 10~12 章，并负责了各款 iOS 硬件环境下的代码验证和调试。其他参与编写的人员有陈超、陈锴、陈佩霞、陈锐、黎华、李鹏钦、李森、李奕辉、李玉莉、刘仲义、卢香清、鲁木应、马向东、麦廷琮、米永刚、欧阳昉、綦彦臣、冉卫华、宋永强、滕科平、王秀丽、王玉芹、魏莹、魏宗寿、温本利。

虽然笔者对本书中所述内容都尽量核实，并多次进行文字校对，但因时间所限，可能还存在疏漏和不足之处，恳请读者批评指正。

编者

目 录

第 1 章 图形图像（一）	1
实例 1 图片浏览器	1
实例 2 疯狂成语	6
实例 3 猜老猫	12
实例 4 图片编辑器	17
实例 5 湖中倒影	22
实例 6 颈部运动	25
实例 7 翻翻看	31
实例 8 节气歌	36
实例 9 行走的青蛙	40
实例 10 变脸	44
实例 11 阴影的变化	46
实例 12 字体下载	48
实例 13 迷雾重重	54
实例 14 重见天日	62
实例 15 眼力测试	69
实例 16 变化的方阵	76
实例 17 调色板	79
实例 18 量尺	83
实例 19 一笔画解答	85
实例 20 公主逃亡记	92
第 2 章 图形图像（二）	98
实例 21 简易相框	98
实例 22 图像滤镜	102
实例 23 图像的点击放大	106
实例 24 万花筒	109
实例 25 浏览商品图片	112
实例 26 具有放大镜的图像	117
实例 27 照片墙	120
实例 28 图像对比	124
实例 29 刮刮卡	129
实例 30 GIF 图像的显示	134

实例 31	评分控件	135
实例 32	图像的多点点击	142
实例 33	裁剪图像	146
实例 34	图像主要颜色的提取	153
实例 35	动物连连看	158
实例 36	人脸识别	171
实例 37	逐层刷新图像	178
实例 38	涂鸦	181
实例 39	图像的 3D 效果浏览	187
第 3 章	图表	193
实例 40	饼状图	193
实例 41	柱状图	198
实例 42	折线图	214
实例 43	波形图	217
实例 44	油量表	221
第 4 章	动画	229
实例 45	飘落的雪花	229
实例 46	自动旋转的太极	231
实例 47	礼花效果	234
实例 48	物理引擎——掉落的蘑菇	236
实例 49	物理引擎——橡皮筋	239
实例 50	吃豆豆	243
实例 51	打砖块	246
实例 52	碰撞的火球	253
实例 53	旋转的滚珠	255
实例 54	永不消失的电波	260
实例 55	牛顿摆	264
实例 56	摇骰子	269
实例 57	计数器	275
实例 58	网格动画	279
实例 59	钟表	283
实例 60	点赞的效果	288
第 5 章	网页视图	295
实例 61	紧急求救中心	295
实例 62	常用网址大全	297
实例 63	改变网页视图中字体的大小	301
实例 64	网页视图的背景透明化	303
实例 65	网页的下拉刷新	305

实例 66	天气预报	312
实例 67	城市地理信息查询	316
实例 68	滑动网页时, 隐藏工具栏	320
实例 69	网页浏览器	322
第 6 章	地图	334
实例 70	地图切换器	334
实例 71	温度带换算器	336
实例 72	地图导航	339
实例 73	位置跟踪器	343
实例 74	指南针	345
实例 75	驴友历程	347
实例 76	地图的位置查找	351
实例 77	3D 地图	353
实例 78	旋转的地图	355
实例 79	行车路线导航	359
实例 80	时区换算器	364
实例 81	自定义地图的标注	368
实例 82	自定义的地图	375
第 7 章	音频和视频	388
实例 83	小钢琴	388
实例 84	手机铃声变化器	392
实例 85	十种语言	396
实例 86	播放歌曲的同时显示歌词	399
实例 87	录音机	405
实例 88	获取系统中所有的音频文件	408
实例 89	讯飞识别	411
实例 90	音乐播放器	415
实例 91	视频播放器	421
第 8 章	内置的应用程序	424
实例 92	工作日计算器	424
实例 93	短信发送	428
实例 94	日历	433
实例 95	添加录	446
第 9 章	触摸和手势	449
实例 96	打地鼠	449
实例 97	人鱼公主换发记	451
实例 98	被挤扁的气球	456
实例 99	撕裂图像	458

实例 100	一个手指实现缩放	462
实例 101	仿小米手机的解锁功能	466
实例 102	QQ 的解锁功能	472
实例 103	拖动选择图片墙	480
第 10 章	照片库与相机	485
实例 104	更改应用程序的背景	485
实例 105	自定义相机	489
实例 106	狙击枪	494
实例 107	水印相机	498
实例 108	QQ 聊天视频效果	503
实例 109	iOS 7 手电筒实现	506
实例 110	三连拍	508
实例 111	条形码/二维码的扫描	514
实例 112	魔术	522
实例 113	录像机	528
第 11 章	传感器	532
实例 114	手机水平放置的测试	532
实例 115	加速的小球	535
实例 116	摇一摇音乐播放器	539
实例 117	根据手机转动显示图像	543
第 12 章	网络	550
实例 118	手机号码查询	550
实例 119	在 Safari 中打开 URL	556
实例 120	后台下载测试	559
实例 121	图像下载队列控制器	564

第 1 章 图形图像（一）

在 iOS 应用程序中，用户都会看到各式各样的图形图像。图形图像可以使用户界面变得丰富多彩，同时可以吸引用户的眼球。本章将主要讲解有关图形图像的一些相关实例。

实例 1 图片浏览器

【实例描述】

本实例实现的功能是图片浏览器。当用户选择界面上的某一图片，单击后，就会进行相应的图片浏览界面。运行效果如图 1.1 所示。



图 1.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“图片浏览器”。
 - (2) 添加图片 1.jpg、2.jpg、3.jpg、4.jpg、5.jpg、6.jpg、7.jpg、8.jpg、9.jpg 到创建项目的 Supporting Files 文件夹中。
 - (3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.2 所示。
- 需要添加的视图、控件以及对它们的设置如表 1-1 所示。

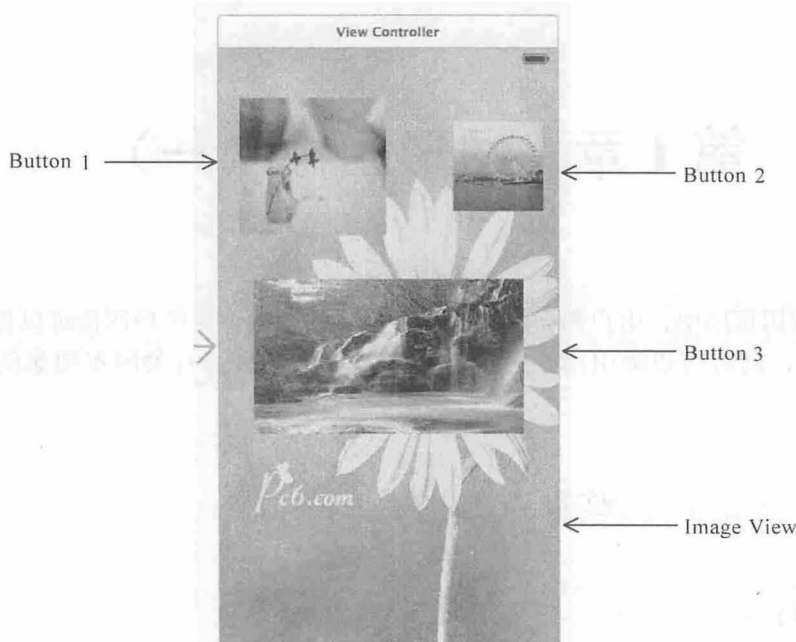


图 1.2 View Controller 视图控制器的设计界面效果

表 1-1 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 7.jpg	
Button1	Title: (空) Background: 8.jpg	将此按钮和 ViewController.h 文件进行动作 aa1:的声明和关联
Button2	Title: (空) Background: 2.jpg	将此按钮和 ViewController.h 文件进行动作 bb:的声明和关联
Button3	Title: (空) Background: 4.jpg	将此按钮和 ViewController.h 文件进行动作 cc:的声明和关联

(4) 创建一个基于 UIViewController 类的 aaViewController 类。

(5) 打开 aaViewController.h 文件，编写代码，实现插座变量、动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface aaViewController : UIViewController{
    IBOutlet UIImageView *iv;           //声明关于图像视图的插座变量
    IBOutlet UIPageControl *pageControl; //声明关于页面控件的插座变量
}
- (IBAction)change:(id)sender;        //id 动作
@end
```

(6) 回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中，将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 aaa，选中 Use Storyboard ID 复选框。

(7) 对 Aa View Controller 视图控制器的设计界面进行设计, 效果如图 1.3 所示。

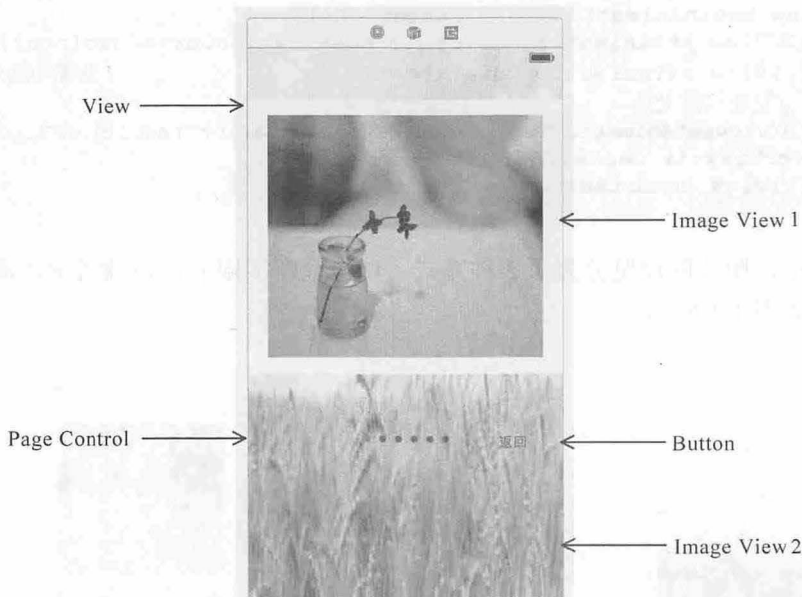


图 1.3 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-2 所示。

表 1-2 视图、控件设置

视图、控件	属性设置	其他
Image View1	Image: 8.jpg	与插座变量iv关联
View		
Image View2	Image: 9.jpg	
Page Control	Pages of Pages: 6 Tint Color: 红色 Current Page: 蓝色	与插座变量pageControl关联 与动作change:关联
Button	Title: 返回	将此按钮和View Controller视图控制器的设计界面关联

(8) 打开 aaViewController.m 文件, 编写代码, 实现图片的浏览。程序代码如下:

```
- (IBAction)change:(id)sender {
    NSInteger next=[pageControl currentPage];           //获取当前的页
    //判断当前的页是否为 0
    if(next==0){
        [iv setImage:[UIImage imageNamed:@"8.jpg"]];
    }else if (next==1){                                //判断当前的页是否为 1
        [iv setImage:[UIImage imageNamed:@"2.jpg"]];
    }else if (next==2){                                //判断当前的页是否为 2
        [iv setImage:[UIImage imageNamed:@"3.jpg"]];
    }else if (next==3){                                //判断当前的页是否为 3
        [iv setImage:[UIImage imageNamed:@"4.jpg"]];
    }else if (next==4){                                //判断当前的页是否为 4
        [iv setImage:[UIImage imageNamed:@"5.jpg"]];
    }else{
        [iv setImage:[UIImage imageNamed:@"6.jpg"]];    //设置图像视图显示的图像
    }
}
```

```

    }
    //实现向左旋转的过渡动画
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1]; //设置动画所需时间
    //设置过渡动画
    [UIView setAnimationTransition:UIViewAnimationTransitionFlipFromLeft
    forView:iv cache:YES];
    [UIView commitAnimations];
}

```

在本实例中，图片的浏览分为了3种方式。由于篇幅的限制，请读者参考源代码。最终的画布效果如图1.4所示。

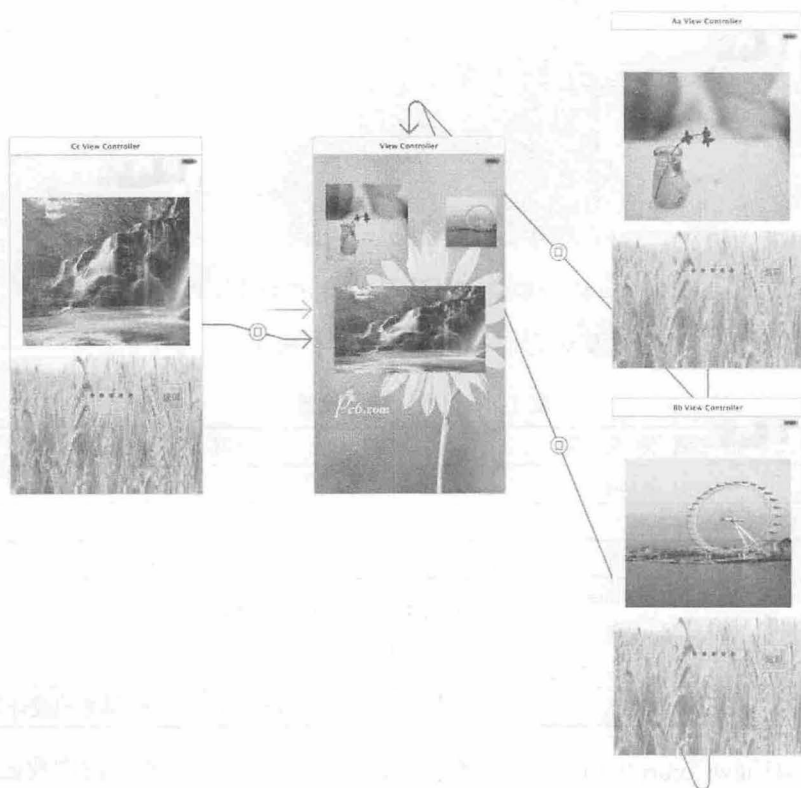


图 1.4 画布效果

(9) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量等的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "aaViewController.h" //头文件
#import "bbViewController.h"
#import "ccViewController.h"
@interface ViewController : UIViewController{
    //声明插座变量
    IBOutlet UIButton *b1;
    IBOutlet UIButton *b2; //关于按钮的插座变量
    IBOutlet UIButton *b3;
    //声明对象
}

```



```

aaViewController *aaaa;
bbViewController *bbbb;
ccViewController *cccc;
}
//动作
- (IBAction)aal:(id)sender;
- (IBAction)bb:(id)sender;
- (IBAction)cc:(id)sender;
@end

```

(10) 在 View Controller 视图控制器的设计界面中，将视图、控件和 ViewController.h 文件声明的插座变量进行关联，如表 1-3 所示。

表 1-3 插座变量的关联

插座变量	关联的视图、控件
b1	与Button1关联
b2	与Button2关联
b3	与Button3关联

(11) 打开 ViewController.h 文件，编写代码，实现在单击按钮后，出现相应的界面。使用的方法如表 1-4 所示。

表 1-4 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
show1	显示Button1按钮
show2	显示Button2按钮
show3	显示Button3按钮
aa:	单击Button1按钮，此按钮放大
aal	以动画的形式到达指定的控制器的视图上
bb:	单击Button2按钮，此按钮放大
bb2	以动画的形式到达指定的控制器的视图上
cc:	单击Button3按钮，此按钮放大
cc2	以动画的形式到达指定的控制器的视图上

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewDidLoad 方法实现了对 3 个按钮的隐藏。程序代码如下：

```

- (void)viewDidLoad
{
    //隐藏按钮
    [b1 setHidden:YES];
    [b2 setHidden:YES];
    [b3 setHidden:YES];
    b2.transform=CGAffineTransformMakeRotation(60*3.14159/180); //旋转按钮
    [self performSelector:@selector(show1) withObject:self afterDelay:1];
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

```

aa:方法实现单击 Button1 按钮后，此按钮放大，并且调用另一个方法。程序代码如下：

```

- (IBAction)aa:(id)sender {
    b1.transform=CGAffineTransformMakeScale(1.2, 1.2);           //放大
    [self performSelector:@selector(aal) withObject:self afterDelay:0.2];
}

```

aal 方法实现以动画的形式到达指定的控制器的视图上。程序代码如下：

```

- (void) aal{
    aaaa=[self.storyboard instantiateViewControllerWithIdentifier:@"aaa"];
    CATransition *t=[CATransition animation];
    t.duration=1;                                           //设置动画所需时间
    t.type=kCATransitionMoveIn;                             //设置动画类型
    t.subtype=kCATransitionFromTop;                         //设置方向
    [self.view addSubview:aaaa.view];
    [self.view.layer addAnimation:t forKey:nil];
}

```

【代码解析】

本实例关键功能是当前页的获取。下面就是这个知识点的详细讲解。

如果想要获取页面控件中当前页的获取，需要使用 `UIPageControl` 的 `currentPage` 属性。其语法形式如下：

```
@property(n nonatomic) NSInteger currentPage;
```

在此代码中就是使用了 `currentPage` 属性来获取当前页并实现图片浏览功能的。代码如下：

```

NSInteger next=[pageControl currentPage];
if(next==0){
    [iv setImage:[UIImage imageNamed:@"8.jpg"]];
}else if (next==1){
    [iv setImage:[UIImage imageNamed:@"2.jpg"]];
}
.....
else{
    [iv setImage:[UIImage imageNamed:@"6.jpg"]];
}

```

实例 2 疯狂成语

【实例描述】

本实例实现的功能是看图猜成语。当单击按钮后，就会出现一个图片，对应图片的内容可以输入相应的成语。当输入正确的成语后，就会进入下一关，直到最后一关。将最后一关也输入了正确的成语后，就会进入到选择关卡界面中。在这里，可以选择任意的关卡进行闯关。运行效果如图 1.5 所示。

【实现过程】

当用户在文本框中输入内容后，单击屏幕，进行字符串的判断。具体的实现步骤如下：

(1) 创建一个项目，命名为“疯狂成语”。

(2) 添加图片 1.png、2.png、3.png、4.png、5.jpg、6.jpg、7.jpg、8.jpg、9.jpg、10.jpg、11.jpg、12.jpg、13.jpg、14.jpg、15.png 到创建项目的 Supporting Files 文件夹中。

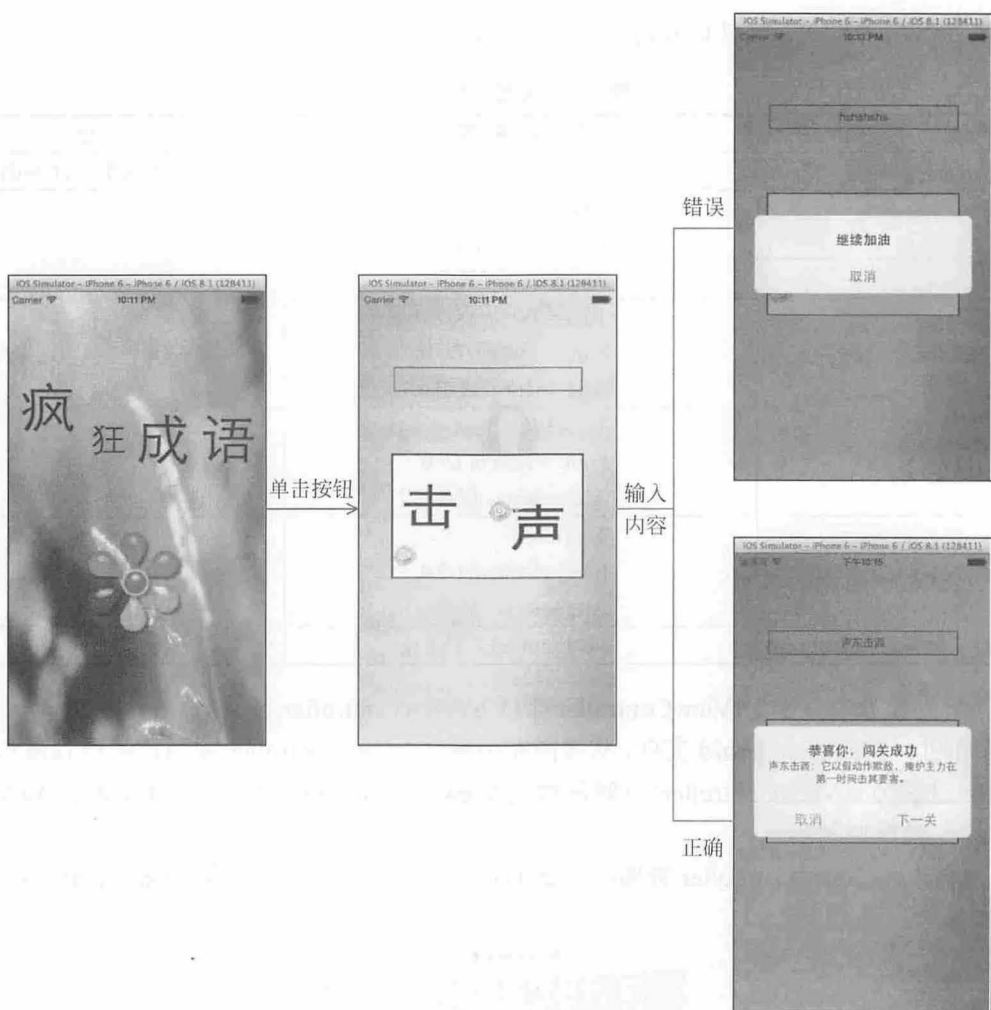


图 1.5 运行效果

(3) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 1.6 所示。

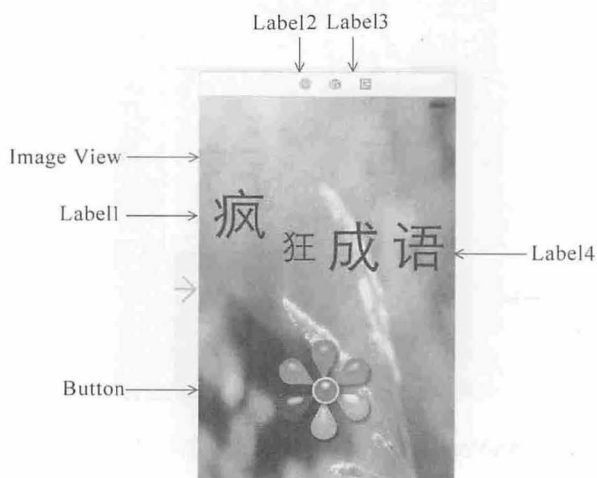


图 1.6 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-5 所示。

表 1-5 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 8.jpg	大小覆盖这个设计界面
Label1	Text: 疯 Font: System 69.0 Alignment: 居中	
Label2	Text: 狂 Font: System 69.0 Alignment: 居中	
Label3	Text: 成 Font: System 69.0 Alignment: 居中	
Label4	Text: 语 Font: System 69.0 Alignment: 居中	
Button	Background: 15.png	

(4) 创建一个基于 UIViewController 类的 aaViewController 类。

(5) 回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。

(6) 对 Aa View Controller 视图控制器的设计界面进行设计，效果如图 1.7 所示。

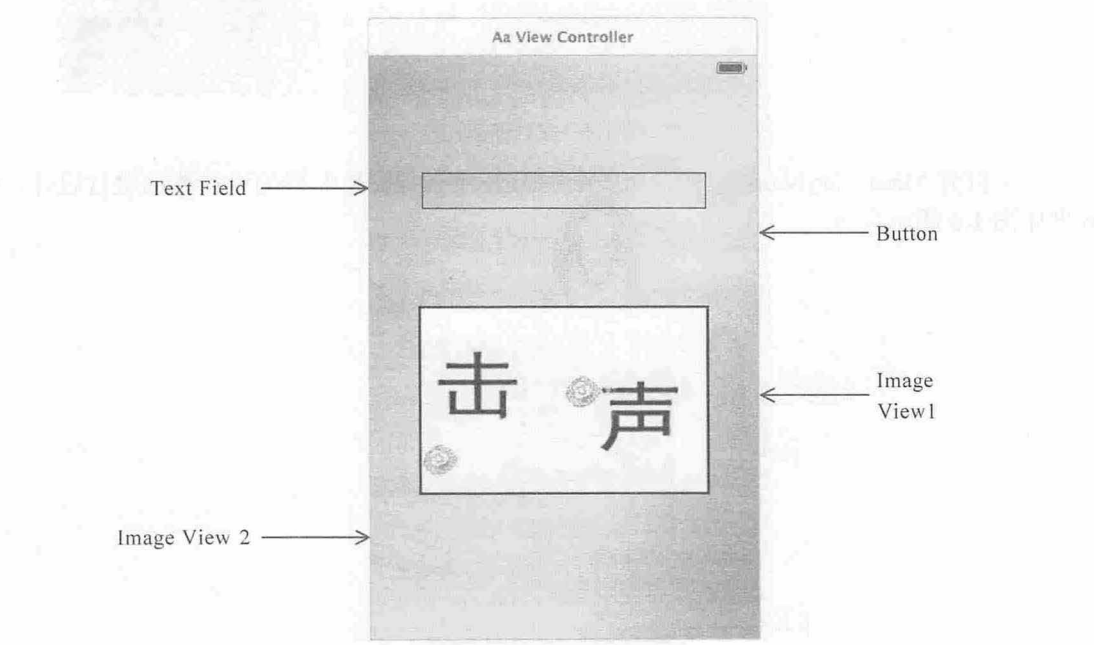


图 1.7 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-6 所示。

表 1-6 视图、控件设置

视图、控件	属性设置	其他
Image View1	Image: 9.jpg	大小覆盖这个设计界面
Button	Title: (空)	将此按钮和aaViewController.h文件进行动作aa:的声明和关联
Text Field	Alignment: 居中 Border Style: 线框	
Image View2	Image: 10.jpg	

(7) 创建一个基于 UIViewController 类的 bbViewController 类。

(8) 打开 aaViewController.h 文件, 编写代码, 实现头文件、对象以及插座变量的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import "bbViewController.h"
@interface aaViewController : UIViewController{
    IBOutlet UITextField *tf;                //声明关于文本框的插座变量
    bbViewController *bb;                    //声明对象
}
- (IBAction)aa:(id)sender;                 //动作
@end
```

(9) 在 Aa View Controller 视图控制器的设计界面中, 将文本框控件和 aaViewController.h 文件中声明的插座变量 tf 进行关联。

(10) 打开 aaViewController.m 文件, 实现对文本框中输入文件的判断以及警告视图的响应。程序代码如下:

```
//单击按钮实现文本框控件中的内容判断
- (IBAction)aa:(id)sender {
    [tf resignFirstResponder];
    //判断文本框中的文本内容是否为"声东击西"
    if([tf.text isEqualToString:@"声东击西"]){
        UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"恭喜你, 闯关成功" message:@"声东击西: 它以假动作欺敌, 掩护主力在第一时间击其要害。" delegate:
            self cancelButtonTitle:@"取消" otherButtonTitles:@"下一关", nil];
        //创建警告视图
        [alert show];
    }else{
        UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"继续加油"
            message:nil delegate:self cancelButtonTitle:@"取消" otherButtonTitles:
            nil];
        [alert show];
    }
}

//实现警告视图中按钮的响应
-(void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTitleAtIndex:buttonIndex];
    //判断 str 中的字符串是否为下一关
    if([str isEqualToString:@"下一关"]){
        CATransition *t=[CATransition animation];
        t.duration=2;                                //设置动画所需时间
        t.subtype=kCATransitionFromBottom;
        t.type=kCATransitionMoveIn;
    }
}
```

```

        bb=[self.storyboard instantiateViewControllerWithIdentifier:@"bb"];
        [self.view addSubview:bb.view];           //添加视图对象
        [self.view.layer addAnimation:t forKey:nil];
    }
}

```

以上就将疯狂成语的第一个关卡设置好了，后面还有 3 个关卡，它们的设置过程都是一样的，由于篇幅的限制，对于设计界面的设计以及代码请读者参考源代码。下面将实现当用户将所有的关卡成功闯关后关卡的选择功能。

（11）创建一个基于 UIViewController 类的 eeViewController 类。

（12）回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 eeViewController。这时新增的 View Controller 视图控制器就变为了 Ee View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 ee，选中 Use Storyboard ID 复选框。

（13）对 Ee View Controller 视图控制器的设计界面进行设计，效果如图 1.8 所示。

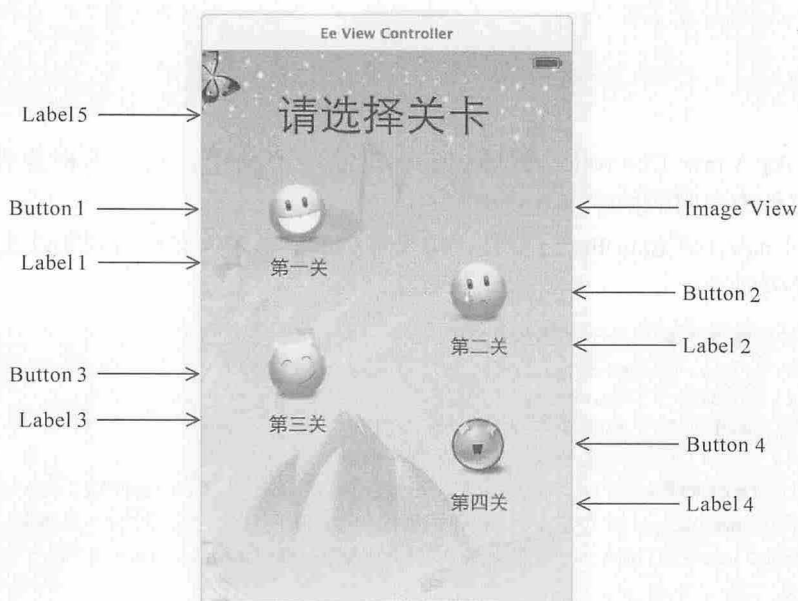


图 1.8 Ee View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-7 所示。

表 1-7 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 14.jpg	
Label1	Text: 第一关	
Button1	Title: (空) Image: 1.png	
Label2	Text: 第二关	
Button2	Image: 2.png	
Label3	Text: 第三关	

续表

视图、控件	属 性 设 置	其 他
Button3	Image: 3.png	
Label4	Text: 第四关	
Button4	Image: 4.png	
Label5	Text: 请选择关卡 Font: System 37.0 Alignment: 居中	

（14）在 Ee View Controller 视图控制器的设计界面中，将按钮控件和其他的视图控制器的设计界面进行关联，如表 1-8 所示。

表 1-8 按钮控件和其他的视图控制器的设计界面的关联

按 钮	关联的视图控制器的设计界面
Button1	Aa View Controller视图控制器的设计界面进行关联
Button2	Bb View Controller视图控制器的设计界面进行关联
Button3	Cc View Controller视图控制器的设计界面进行关联
Button4	Dd View Controller视图控制器的设计界面进行关联

（15）在 View Controller 视图控制器中，按钮和 Aa View Controller 视图控制器的设计界面进行关联，这时画布效果如图 1.9 所示。



图 1.9 画布效果

【代码解析】

本实例关键功能是在字符串的比较。下面就是这个知识点的详细讲解。
在本实例中，字符串的比较使用了 NSString 的 isEqualToString:方法，它的功能是判断

字符串是否相等。代码如下：

```
if([tf.text isEqualToString:@"声东击西"]){
    .....
}else{
    .....
}
```

实例3 猜 老 猫

【实例描述】

本实例实现的功能是猜老猫的游戏。当用户单击按钮，1 秒后出现的牌数。这时，会判断选择的按钮是否和牌数一样。如果一样，就会弹出游戏成功的警告视图；如果失败，就会弹出游戏失败的警告视图。运行效果如图 1.10 所示。

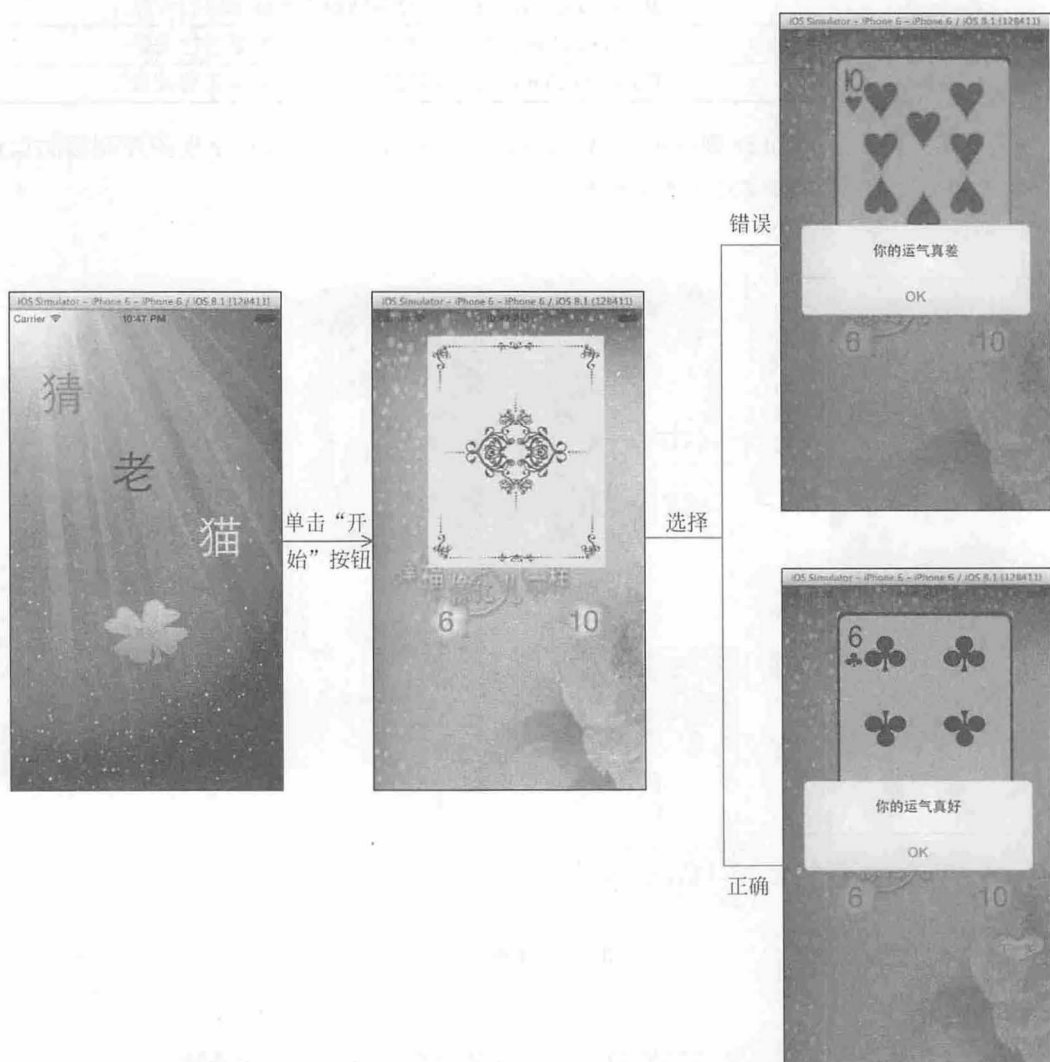


图 1.10 运行效果

【实现过程】

(1) 创建一个项目，命名为“猜老猫”。

(2) 添加图片 1.png、2.png、3.png、4.jpg、5.jpg、6.jpg、7.jpg、8.jpg、9.jpg、10.jpg、11.jpg、12.jpg、13.jpg 添加到创建项目的 Supporting Files 文件夹中。

(3) 单击打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.11 所示。

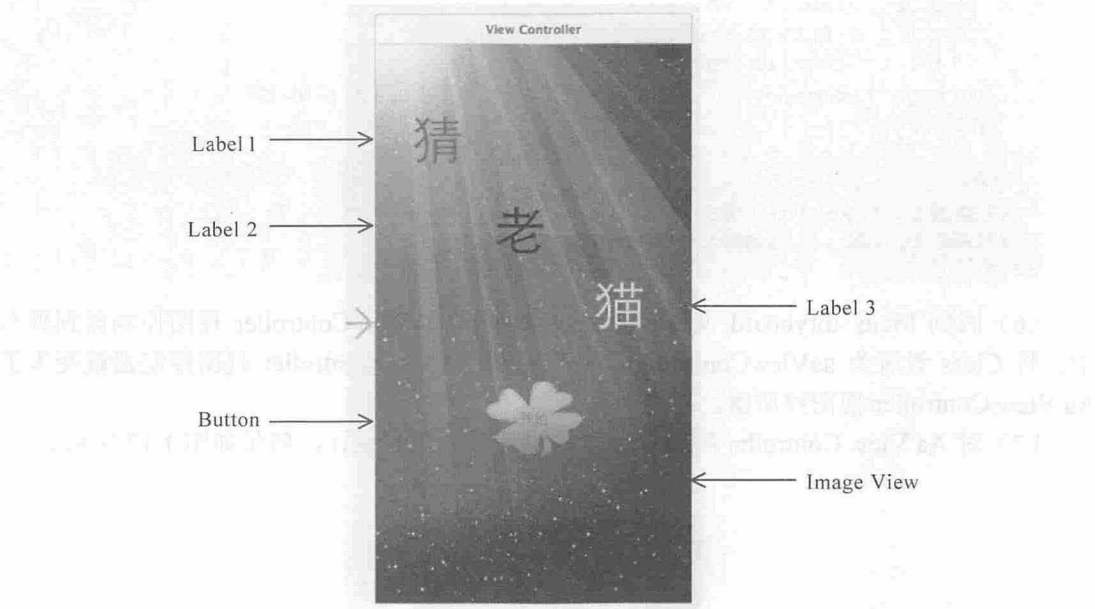


图 1.11 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-9 所示。

表 1-9 视图、控件设置

视图、控件	属性设置	其他
Image View	Image: 8.jpg	将其覆盖整个设计界面
Label1	Text: 猜 Color: 红色 Font: System 53.0 Alignment: 居中	
Label2	Text: 老 Color: 黑色 Font: System 53.0 Alignment: 居中	
Label3	Text: 猫 Color: 黄色 Font: System 53.0 Alignment: 居中	
Button	Title: 开始 Background: 3.png	

(4) 创建一个基于 UIViewController 类的 aaViewController 类。

(5) 打开 aaViewController.h 文件，编写代码，实现插座变量、对象以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface aaViewController : UIViewController{
    IBOutlet UIImageView *imv;                //声明关于图像视图的插座变量
    //声明关于按钮的插座变量
    IBOutlet UIButton *bu1;
    IBOutlet UIButton *bu2;
    IBOutlet UIButton *bu3;
    UIImage *image;                          //声明对象
    NSString *s;
}
//动作
- (IBAction)aa:(id)sender;
- (IBAction)bb:(id)sender;
@end
```

(6) 回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。

(7) 对 Aa View Controller 视图控制器的设计界面进行设计，效果如图 1.12 所示。

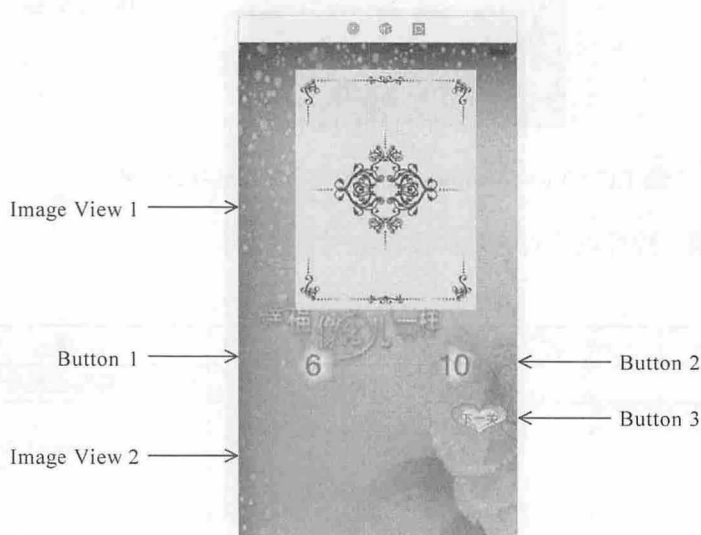


图 1.12 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-10 所示。

表 1-10 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View1	Image: 6.jpg	与插座变量imv关联
Image View2	Image: 7.jpg	
Button1	Title: 6 Font: System 34.0 Background: 1.png	与插座变量bu1关联 与动作aa:关联

续表

视图、控件	属性设置	其他
Button2	Title: 10 Font: System 34.0 Background: 1.png	与插座变量bu2关联 与动作bb:关联
Button3	Title: 下一关 Background: 2.png	

(8) 将开始按钮和 Aa View Controller 视图控制器的设计界面进行关联。

(9) 打开 aaViewController.m 文件, 编写代码, 实现判断单击的按钮是否和图片中出现的数字一样。使用的方法如表 1-11 所示。

表 1-11 aaViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
aa:	单击Button1按钮
aa1	旋转的动画效果
aa2	判断并赋值
aa3	判断并显示警告视图
bc	旋转
bb:	单击Button2按钮

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中, viewDidLoad 方法实现 Image View1 中图像的随机出现。程序代码如下:

```

- (void)viewDidLoad
{
    [bu3 setHidden:YES];
    int i=arc4random()%2;           //产生随机数
    switch (i) {
        case 0:
            image=[UIImage imageNamed:@"10.jpg"]; //创建图像对象
            break;
        default:
            image=[UIImage imageNamed:@"9.jpg"]; //创建图像对象
            break;
    }
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

```

aa1 方法实现图像视图的旋转功能。程序代码如下:

```

- (void)aa1{
    [UIView beginAnimations:@" " context:nil];
    imv.image=image; //设置图像视图显示的图像
    [UIView setAnimationDuration:3]; //设置动画所需时间
    [UIView setAnimationTransition:UIViewAnimationTransitionFlipFromLeft
    forView:imv cache:YES];
    [UIView commitAnimations];
    [self performSelector:@selector(aa2) withObject:self afterDelay:1];
}

```

aa2 方法实现判断图像视图显示的图像并且为 s 进行赋值，程序代码如下：

```
-(void)aa2{
    //判断图像视图对象 imv 显示的图像是否为 9.jpg
    if([imv.image isEqual:[UIImage imageNamed:@"9.jpg"]]){
        s=@"6";
    }else if([imv.image isEqual:[UIImage imageNamed:@"10.jpg"]]){
        s=@"10";
    }
    [self performSelector:@selector(aa3) withObject:self afterDelay:1];
    //经过 1 秒后执行 aa3 方法
}
```

aa3 方法将按钮中的文本内容和 s 进行比较。程序代码如下：

```
-(void)aa3{
    if([bu1.isHidden]==NO){
        //判断按钮的标题是否为 s 字符串中的内容
        if([bu1.titleLabel.text isEqualToString:s]){
            UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"你的运气真好" message:@"" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
            [alert show];
            [bu3 setHidden:NO];
        }else{
            UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"你的运气真差" message:@"" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
            [alert show];
            //创建警告视图
        }
    }else{
        //判断按钮的标题是否为 s 字符串中的内容
        if ([bu2.titleLabel.text isEqualToString:s]){
            UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"你的运气真好" message:@"" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
            [alert show];
            [bu3 setHidden:NO];
        }else{
            UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"你的运气真差" message:@"" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
            [alert show];
            //创建警告视图
        }
    }
    [self performSelector:@selector(bc) withObject:self afterDelay:1];
    //经过 1 秒后执行 bc 方法
}
```

在本实例中，猜老猫游戏分为了 2 关，每一关的难易程度是不一样的。由于篇幅的限制，对于设计界面的设计以及代码请读者参考源代码。

【代码解析】

本实例关键功能是在显示图像与选择按钮的判断。下面就是这个知识点的详细讲解。

本实例中要实现显示图像与选择按钮的判断，首先，要对图像的显示内容进行判断，然后对 s 进行赋值，代码如下：

```
if(imv.image==[UIImage imageNamed:@"9.jpg"]){
```



```
s=@"6";  
}else if(imv.image==[UIImage imageNamed:@"10.jpg"]){  
    s=@"10";  
}
```

然后，将按钮的标题与 s 进行判断，实现了显示图像与选择按钮的判断，代码如下：

```
if([bul.titleLabel.text isEqualToString:s]){  
    .....  
}else{  
    .....  
}
```

实例4 图片编辑器

【实例描述】

本实例主要的功能是利用对 Image View 图像视图的放大、缩小、平移等，实现一个图片编辑器。在此编辑器中，不仅可以切换背景、装饰，还可以对装饰进行编辑。运行效果如图 1.13 所示。

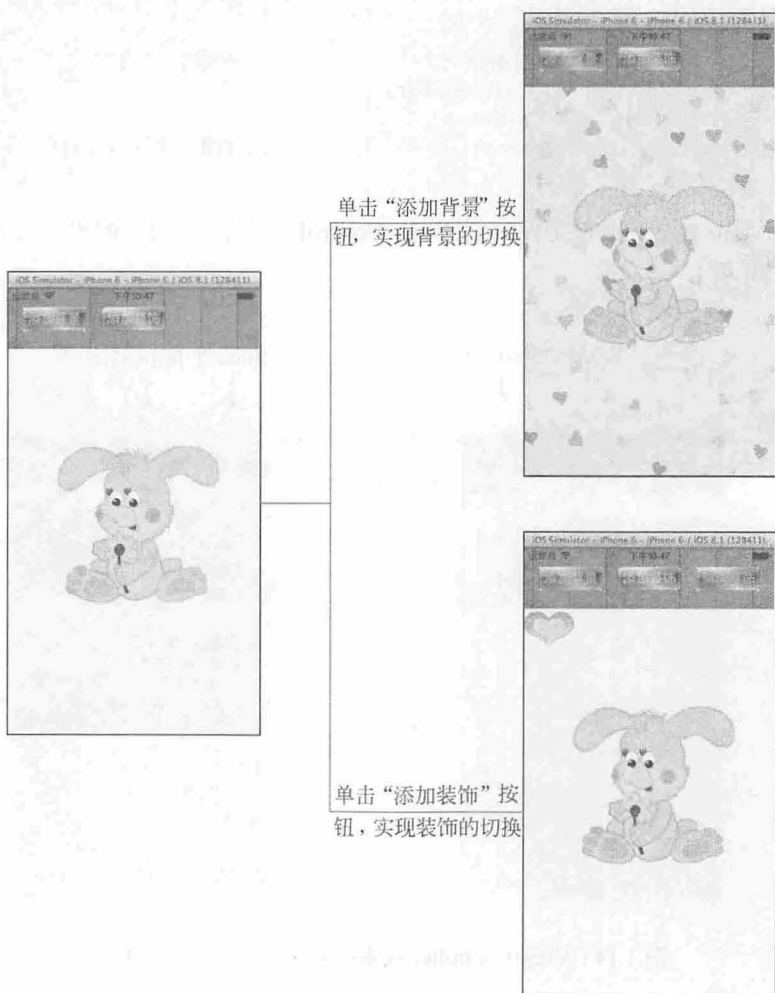


图 1.13 运行效果

【实现过程】

(1) 创建一个项目，命名为“图片编辑器”。

(2) 添加图片 1.jpg、2.png、3.png、4.png、5.png、6.png、7.jpg、8.jpg、9.png、10.png、11.png、12.jpg 到创建项目的 Supporting Files 文件夹中。

(3) 打开 ViewController.h 文件中，编写代码，实现插座变量、实例变量、对象、动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController<UIActionSheetDelegate, UIAlertView
Delegate>{
    //声明插座变量
    IBOutlet UIImageView *iv1;
    IBOutlet UIView *vv;
    IBOutlet UIButton *button;
    UIImageView *iv2;
    float x,y,i;
}
//动作
- (IBAction)aa:(id) sender;
- (IBAction)bb:(id) sender;
- (IBAction)cc:(id) sender;
- (IBAction)scalebig:(id) sender;
- (IBAction)scalelittle:(id) sender;
- (IBAction)doun:(id) sender;
- (IBAction)right:(id) sender;
@end
```

//声明对象
//声明实例
//单击 Button1 按钮后，弹出动作表单
//单击 Button2 按钮后，弹出警告视图
//图像放大动作
//图像向右移动动作

(4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.14 所示。

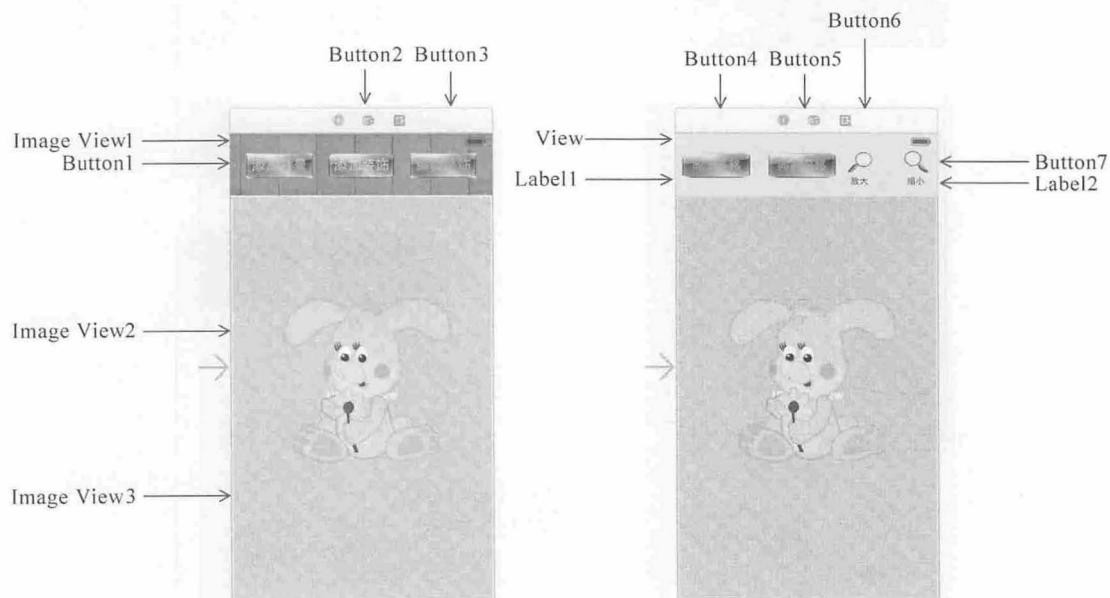


图 1.14 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-12 所示。

表 1-12 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View1	Image: 12.jpg	
Button1	Title: 添加背景 Font: System Bold 17.0 Text Color: 绿色 Background: 1.jpg	与动作aa:关联
Button2	Title: 添加装饰 Font: System Bold 17.0 Text Color: 黄色 Background: 2.png	与动作bb:关联
Button3	Title: 编辑装饰 Font: System Bold 17.0 Text Color: 绿色 Background: 1.jpg	与插座变量button关联 与动作cc:关联
Image View3		与插座变量iv1关联
Image View2	Image: 9.png	
View	Background: 粉色	与插座变量vv关联
Button4	Title: 向下移动 Font: System Bold 15.0 Text Color: 绿色 Background: 1.jpg	与动作down:关联
Button5	Title: 向右移动 Font: System Bold 15.0 Text Color: 绿色 Background: 1.jpg	与动作right:关联
Button6	Title: （空） Background: 6.png	与动作scalebig:关联
Label1	将Text: 放大 Font: System 11.0 Alignment: 居中	
Button7	Title: （空） Background: 5.png	与动作scalelittle:关联
Label2	Text: 缩小 Font: System 11.0 Alignment: 居中	

（5）打开 ViewController.m 文件，编写代码，实现背景、装饰的切换，以及对装饰的编辑功能。使用的方法如表 1-13 所示。

表 1-13 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
aa:	单击Button1按钮后，弹出动作表单
bb:	单击Button2按钮后，弹出警告视图

续表

方 法	功 能
actionSheet:clickedButtonAtIndex:	动作表单的响应
alertView:clickedButtonAtIndex:	警告视图的响应
cc:	显示View
scalebig:	图像的放大
scalelittle:	图像的缩小
down:	图像的向下移动
right:	图像的向右移动

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，actionSheet:clickedButtonAtIndex:方法实现对动作表单的响应。程序代码如下：

```
-(void)actionSheet:(UIActionSheet *)actionSheet clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[actionSheet buttonTitleAtIndex:buttonIndex];
    //判断 str 是否与"背景 2"相等
    if([str isEqualToString:@"背景 1"]){
        iv1.image=[UIImage imageNamed:@"7.jpg"];
    }else if ([str isEqualToString:@"背景 2"]){ //判断 str 是否与"背景 2"相等
        iv1.image=[UIImage imageNamed:@"8.jpg"];
    }else{
        iv1.image=[UIImage imageNamed:@"10.png"]; //设置图像视图显示的图像
    }
}
```

alertView:clickedButtonAtIndex:方法实现警告视图的响应。程序代码如下：

```
-(void>alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTitleAtIndex:buttonIndex];
    if([str isEqualToString:@"装饰 1"]){ //判断 str 是否与"装饰 1"相等
        iv2.image=[UIImage imageNamed:@"3.png"];
        [button setHidden:NO];
    }else if ([str isEqualToString:@"装饰 2"]){ //判断 str 是否与"装饰 2"相等
        iv2.image=[UIImage imageNamed:@"4.png"];
        [button setHidden:NO];
    }else{
        iv2.image=[UIImage imageNamed:@"11.png"]; //设置图像视图显示的图像
        [button setHidden:NO];
    }
}
```

scalebig:方法实现单击放大的按钮后，图像的放大功能。程序代码如下：

```
-(IBAction)scalebig:(id)sender {
    CGRect f=[iv2 frame];
    iv2.frame=f;
    if(i<=2){
        iv2.transform=CGAffineTransformMakeScale(i, i); //缩放
    }
    i+=0.1;
}
```

scalelittle:方法实现单击缩小的按钮后，图像的缩小功能。程序代码如下：

```

- (IBAction)scalelittle:(id)sender {
    if(i>=0.2){
        iv2.transform=CGAffineTransformMakeScale(i, i);           //缩放
    }
    i-=0.1;
}

```

down:方法实现单击“向下移动”按钮后，图像向下移动的功能。程序代码如下：

```

- (IBAction)down:(id)sender {
    if(y<=568){
        iv2.transform=CGAffineTransformMakeTranslation(x, y);      //平移
    }
    y+=20;
}

```

right:方法实现单击“向右移动”按钮后，图像向右移动的功能。程序代码如下：

```

- (IBAction)right:(id)sender {
    CGRect f=[iv2 frame];
    iv2.frame=f;
    if(x<=320){
        iv2.transform=CGAffineTransformMakeTranslation(x,y);      //平移
        x+=10;
    }
}

```

【代码解析】

本实例关键功能是缩放和移动。下面依次讲解这两个知识点。

1. 缩放

缩放需要使用 CGAffineTransform 的 CGAffineTransformMakeScale 函数。其语法形式如下：

```

CGAffineTransform CGAffineTransformMakeScale (
    CGFloat sx,
    CGFloat sy
);

```

其中，CGFloat sx 表示 x 轴的缩放因子；CGFloat sy 表示 y 轴的缩放因子。在本实例中就使用了 CGAffineTransformMakeScale 函数实现了图像的缩放功能。程序代码如下：

```
iv2.transform=CGAffineTransformMakeScale(i, i);
```

其中，第一个 i 表示 x 轴的缩放因子；第二个 i 表示 y 轴的缩放因子。

2. 移动

平移需要使用 CGAffineTransform 的 CGAffineTransformMakeTranslation 函数。其语法形式如下：

```

CGAffineTransform CGAffineTransformMakeTranslation (
    CGFloat tx,
    CGFloat ty
);

```

其中，CGFloat tx 表示 x 轴的移动因子；CGFloat ty 表示 y 轴的移动因子。在本实例中就使用了 CGAffineTransformMakeTranslation 函数实现了图像的向右向下的移动功能。程序代码如下：

```
iv2.transform=CGAffineTransformMakeTranslation(x,y);
```

其中，x 表示 x 轴的移动因子；y 表示 y 轴的移动因子。

实例 5 湖中倒影

【实例描述】

本实例主要实现的功能是在水中实现图像的反射效果，并且可以通过滑块调整反射图片的阴影。运行效果如图 1.15 所示。

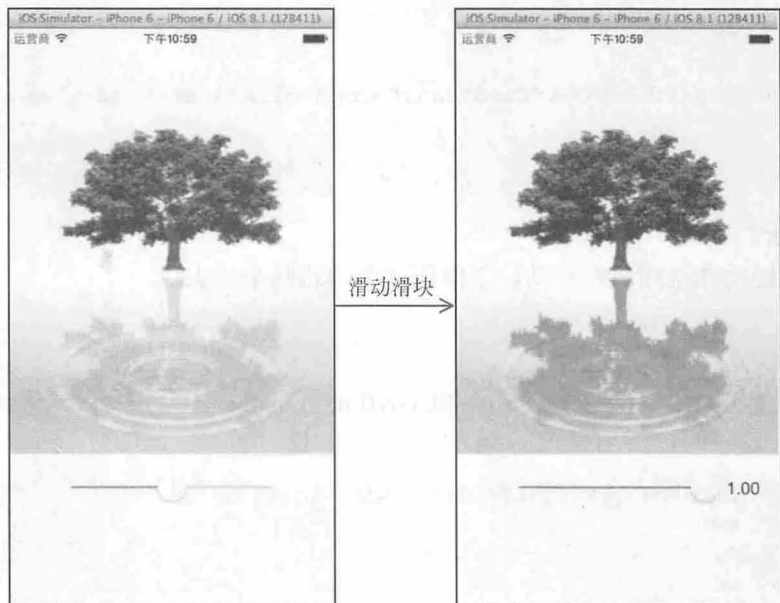


图 1.15 运行效果

【实现过程】

- (1) 创建一个项目，命名为“湖中倒影”。
- (2) 添加图片 1.jpg、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现插座变量、实例变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //声明有关图像视图的插座变量
    IBOutlet UIImageView *iv;
    IBOutlet UIImageView *fv;
    IBOutlet UISlider *slider;
    IBOutlet UILabel *a;
    //声明有关滑块控件的插座变量
```

```
CGFloat alpha;                                //声明实例变量
}
- (IBAction)change:(id)sender;
@end
```

(4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.16 所示。

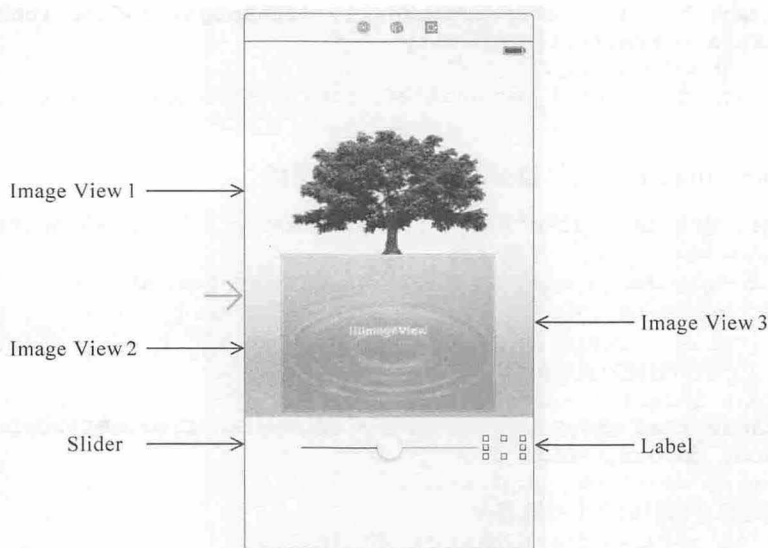


图 1.16 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-14 所示。

表 1-14 视图、控件设置

视图、控件	属性设置	其他
Image View1	Image: 2.png	与插座变量iv关联
Image View2	Image: 1.jpg Alpha: 0.3	
Image View3		与插座变量fv关联
Slider		与插座变量slider关联 与动作change:关联
Label	Text: (空)	与插座变量a关联

(5) 打开 ViewController.m 文件，编写代码，实现倒影的效果。使用的方法如表 1-15 所示。

表 1-15 ViewController.m文件中方法总结

方法	功能
viewDidLoad	视图加载后调用，实现初始化
CreateGradientImage	获取渐变图像
MyCreateBitmapContext	获取位图上下文
reflectedImage:withHeight:	获取倒影
change:	控制倒影的透明度

其中，viewDidLoad 方法实现界面的初始化设置。程序代码如下：


```

- (void)viewDidLoad
{
    self.view.autoresizesSubviews = YES;
    self.view.userInteractionEnabled = YES;
    //确定创建的倒影大小
    NSInteger reflectionHeight = iv.bounds.size.height * kReflection
    Fraction;
    //将创建的倒影放到 fv 图像视图中
    fv.image = [self reflectedImage:iv withHeight:reflectionHeight];
    fv.alpha = kReflectionOpacity;
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

```

CreateGradientImage 方法创建并获取一个渐变图像。

```

CGImageRef CreateGradientImage(int pixelsWide, int pixelsHigh){
    CGImageRef cgimage = NULL;
    CGColorSpaceRef space = CGColorSpaceCreateDeviceGray(); //创建渐变
    CGContextRef gradientBitmapContext = CGContextCreate(NULL,
    pixelsWide, pixelsHigh, 8, 0, space, 0); //创建位图上下文
    //定义开始和结束的灰度值
    CGFloat colors[] = {0.0, 1.0, 1.0, 1.0};
    CGGradientRef grayScaleGradient = CGGradientCreateWithColorComponents
    (space, colors, NULL, 2);
    CGColorSpaceRelease(space);
    //创建梯度向量的开始和结束点
    CGPoint gradientStartPoint = CGPointZero;
    CGPoint gradientEndPoint = CGPointMake(0, pixelsHigh);
    //绘制渐变为灰度的位图上下文
    CGContextDrawLinearGradient(gradientBitmapContext, grayScaleGradient,
    gradientStartPoint, gradientEndPoint, kCGGradientDrawsAfterEndLocation);
    CGGradientRelease(grayScaleGradient);
    cgimage = CGContextCreateImage(gradientBitmapContext);
    CGContextRelease(gradientBitmapContext);
    return cgimage;
}

```

MyCreateBitmapContext 方法创建并获取一个位图上下文。程序代码如下：

```

CGContextRef MyCreateBitmapContext(int pixelsWide, int pixelsHigh){
    CGColorSpaceRef space = CGColorSpaceCreateDeviceRGB(); //创建色彩空间
    CGContextRef bitmapContext = CGContextCreate(NULL, pixelsWide,
    pixelsHigh, 8, 0, space, (kCGBitmapByteOrder32Little | kCGImageAlpha
    Premultiplied First)); //创建位图上下文
    CGColorSpaceRelease(space);
    return bitmapContext;
}

```

reflectedImage:withHeight:方法实现对倒影图像的获取。程序代码如下：

```

- (UIImage *)reflectedImage:(UIImageView *)fromImage withHeight:(NSInteger)
height{
    //判断 height 是否为 0
    if(height == 0){
        return nil;
    }
    //创建位图上下文
    CGContextRef mainViewContentContext = MyCreateBitmapContext(fromImage.
    bounds.size.width, height);
}

```

```

CGImageRef gradientMaskImage = CreateGradientImage(1, height);
CGContextClipToMask(mainViewContentContext, CGRectMake(0.0, 0.0,
fromImage.bounds.size.width, height), gradientMaskImage);
//在图形上下文的指定矩形区域内打上mask
CGImageRelease(gradientMaskImage);
CGContextTranslateCTM(mainViewContentContext, 0.0, height);
//平移
CGContextScaleCTM(mainViewContentContext, 1.0, -1.0); //缩放
//将绘制的图像放到位图上下文中
CGContextDrawImage(mainViewContentContext, fromImage.bounds, fromImage.
image.CGImage);
CGImageRef reflectionImage = CGBitmapContextCreateImage(mainView
ContentContext);
CGContextRelease(mainViewContentContext);
UIImage *theImage = [UIImage imageWithCGImage:reflectionImage];
//实例化图像对象

CGImageRelease(reflectionImage);
return theImage;
}

```

change:方法实现对倒影的控制。程序代码如下:

```

- (IBAction)change:(id)sender {
    slider = (UISlider *)sender;
    CGFloat val = [slider value];
    fv.alpha = val; //设置透明度
    a.text=[NSString stringWithFormat:@"%0.2f",val];
}

```

【代码解析】

本实例关键功能是图像的翻转效果。下面就是这个知识点的详细讲解。

要实现图像的翻转效果,需要使用 CGContext 的 CGContextScaleCTM 函数。其语法形式如下:

```

void CGContextScaleCTM (
    CGContextRef c,
    CGFloat sx,
    CGFloat sy
);

```

其中, CGContextRef c 表示上下文; CGFloat sx 表示 x 轴的缩放因子; CGFloat sy 表示 y 轴的缩放因子。在此代码中,使用了 CGContextScaleCTM 函数实现了图像倒影的效果,代码如下:

```
CGContextScaleCTM(mainViewContentContext, 1.0, -1.0);
```

其中, mainViewContentContext 表示上下文; 1.0 表示 x 轴的缩放因子; 1.0 表示 y 轴的缩放因子。

实例6 颈部运动

【实例描述】

本实例实现的功能实现一个颈部运动的应用程序,此应用可以很好地缓解颈部的疲

劳。运行效果如图 1.17 所示。



图 1.17 运行效果

【实现过程】

- (1) 创建一个项目，命名为“颈部运动”。
- (2) 添加图片 1.pn、2.jpg、3.png、4.png 到创建项目的 Supporting Files 文件夹中。
- (3) 单击打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.18 所示。

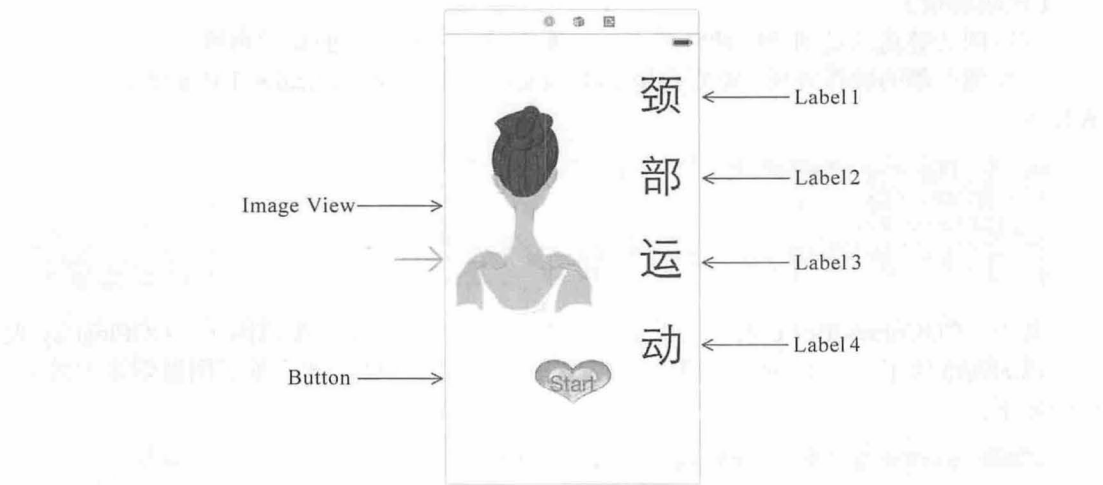


图 1.18 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-16 所示。

表 1-16 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 1.jpg	
Label1	Text: 颈 Font: System 56.0 Alignment: 居中	

续表

视图、控件	属性设置	其他
Label2	Text: 部 Font: System 56.0 Alignment: 居中	
Label3	Text: 运 Font: System 56.0 Alignment: 居中	
Label4	Text: 动 Font: System 56.0 Alignment: 居中	
Button	Title: Start Font: System 28.0 Background: 4.png	

(4) 创建一个基于 UIViewController 类的 aaViewController 类。

(5) 打开 aaViewController.h 文件, 编写代码, 实现插座变量的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface aaViewController : UIViewController{
    //声明有关按钮的插座变量
    IBOutlet UIButton *button1;
    IBOutlet UIButton *button2;
    IBOutlet UIButton *button3;
}
@end
```

(6) 打开 Main.storyboard 文件, 从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。

(7) 对 Aa View Controller 视图控制器的设计界面进行设计, 效果如图 1.19 所示。



图 1.19 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-17 所示。

表 1-17 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅橘黄色	
Label	Text: 请根据脖子的灵活度选择速度: Font: System 56.0 Alignment: 居中	
Button1	Title: 慢 Font: System 20.0 Background: 3.png	将其移到位置为 (-12,190) 的位置上 与插座变量button1关联
Button2	Title: 中 Font: System 20.0 Background: 3.png	将其移到位置为 (328,290) 的位置上 与插座变量button2关联
Button3	Title: 快 Font: System 20.0 Background: 3.png	将其移到位置为 (-12,390) 的位置上 与插座变量button3关联

(8) 单击打开 aaViewController.m 文件, 编写代码, 实现动画效果。使用的方法如表 1-18 所示。

表 1-18 aaViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
aa	Button1按钮的动画效果
bb	Button2按钮的动画效果
bb	Button2按钮的动画效果

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, aa 方法实现使用动画的效果显示 Button1 按钮。程序代码如下:

```
-(void)aa{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1]; //设置动画所需的时间
    button1.frame=CGRectMake(116, 160, 88, 60);
    [UIView commitAnimations];
    [self performSelector:@selector(bb) withObject:self afterDelay:1.2];
    //经过 1.2 秒后执行 bb 方法
}
```

(9) 创建一个基于 UIViewController 类的 bbViewController 类。

(10) 打开 bbViewController.h 文件, 编写代码, 实现插座变量、对象的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface bbViewController : UIViewController{
    UIImageView *imv; //声明对象
    IBOutlet UILabel *label; //声明插座变量
}
```

```
}  
@end
```

(11) 打开 Main.storyboard 文件, 从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 bbViewController。这时, 新增的 View Controller 视图控制器就变为了 Bb View Controller 视图控制器。

(12) 对 Bb View Controller 视图控制器的设计界面进行设计, 效果如图 1.20 所示。

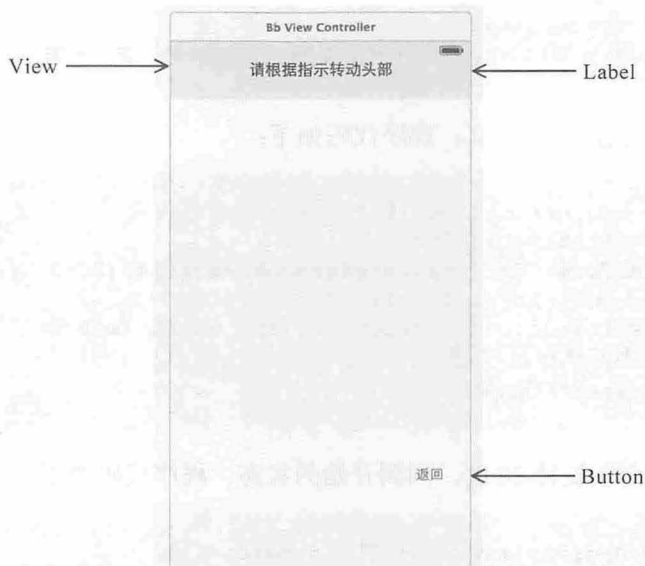


图 1.20 Bb View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-19 所示。

表 1-19 视图、控件设置

视图、控件	属 性 设 置	其 他
View	Background: 黄色	
Label	Text: 请根据指示转动头部 Alignment: 居中	与插座变量label关联
Button	Title: 返回 Font: System 20.0 Background: 3.png	

(13) 打开 bbViewController.m 文件, 编写代码, 实现头部转动的动画。使用的方法如表 1-20 所示。

表 1-20 bbViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
aa	向右旋转 20 度
ab	向左旋转, 回到开始的状态
ac	向左旋转 20 度
ad	向右旋转, 回到开始的状态

其中，viewDidLoad 方法实现对图像视图的创建。程序代码如下：

```
-(void)viewDidLoad
{
    UIImageView *imv=[[UIImageView alloc] initWithFrame:CGRectMake(70, 130, 180, 180)];
    imv.image=[UIImage imageNamed:@"1.png"];           //设置图像视图显示的图像
    [self.view addSubview:imv];
    [self performSelector:@selector(aa) withObject:self afterDelay:3];
    //经过 3 秒后执行 aa 方法
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}
```

aa 方法实现向右旋转 20 度。程序代码如下：

```
-(void)aa{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2];
    imv.transform=CGAffineTransformMakeRotation(20*3.14/180);           //旋转
    [UIView commitAnimations];
    [self performSelector:@selector(ab) withObject:self afterDelay:5];
    //经过 5 秒后执行 ab 方法
    label.text=@"向右转";
}
```

ab 方法实现向左旋转 20 度，回到开始的状态。程序代码如下：

```
-(void)ab{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2];
    imv.transform=CGAffineTransformMakeRotation(0);           //旋转
    [UIView commitAnimations];
    [self performSelector:@selector(ac) withObject:self afterDelay:5];
    //经过 5 秒后执行 ac 方法
    label.text=@"向左转";
}
```

ac 方法实现向左旋转 20 度。程序代码如下：

```
-(void)ac{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2];
    imv.transform=CGAffineTransformMakeRotation(-20*3.14/180);           //旋转
    [UIView commitAnimations];
    [self performSelector:@selector(ad) withObject:self afterDelay:5];
    //经过 5 秒后执行 ad 方法
    label.text=@"向左转";
}
```

在本实例中，颈部运动一共分为 3 个阶段，在每一个阶段，颈部运动的速度会改变。由于篇幅的限制，对于设计界面的设计以及代码请读者参考源代码。

【代码解析】

本实例关键功能是图像的旋转。下面就是这个知识点的详细讲解。

在本实例中，图像的旋转采用的是 CGAffineTransformMakeRotation 函数，代码如下：

```
imv.transform=CGAffineTransformMakeRotation(20*3.14/180);
```

其中，20*3.14/180 表示图像旋转的度数。

实例7 翻 翻 看

【实例描述】

本实例的功能是利用旋转的动画效果，实现在单击两个按钮后，判断出现的图片是否一样。如果一样就两个按钮隐藏；如果不一样，就显示原来的图片。运行效果如图 1.21 所示。

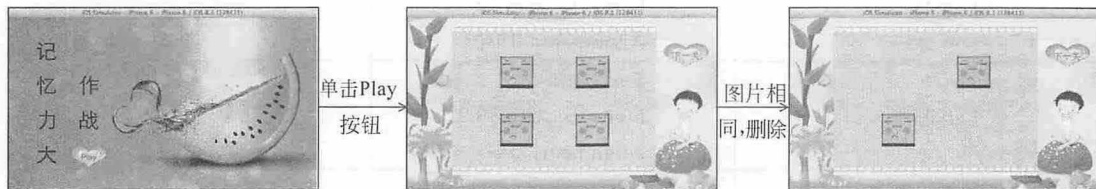


图 1.21 运行效果

【实现过程】

(1) 创建一个项目，命名为“翻翻看”。

(2) 添加图片 1.png、2.jpg、3.jpg、4.jpg、5.jpg、6.jpg、7.jpg、8.jpg、9.jpg、10.jpg、11.jpg 到创建项目的 Supporting Files 文件夹中。

(3) 打开目标窗口，选择 General 选项，在 Deployment Info 面板下，只选中 Landscape Left 复选框和 Landscape Right 复选框。

(4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.22 所示。

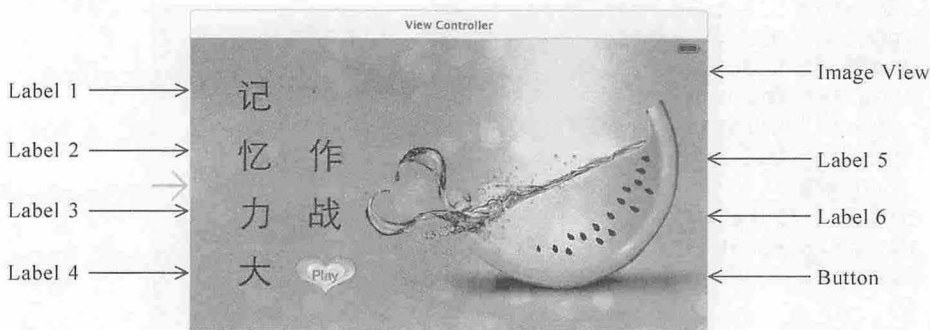


图 1.22 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-21 所示。

表 1-21 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 4.jpg	
Label1	Text: 记 Font: System 39.0 Alignment: 居中	

续表

视图、控件	属 性 设 置	其 他
Label2	Text: 忆 Font: System 39.0 Alignment: 居中	
Label3	Text: 力 Font: System 39.0 Alignment: 居中	
Label4	Text: 大 Font: System 39.0 Alignment: 居中	
Label5	Text: 作 Font: System 39.0 Alignment: 居中	
Label6	Text: 战 Font: System 39.0 Alignment: 居中	
Button	Title: Play Background: 1.png	

(5) 创建一个基于 UIViewController 类的 aaViewController 类。

(6) 打开 aaViewController.h 文件，编写代码，实现插座变量、对象、动作等的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface aaViewController : UIViewController{
    //声明插座变量
    IBOutlet UIButton *b1;
    IBOutlet UIButton *b2;
    IBOutlet UIButton *b3;
    IBOutlet UIButton *b4;
    IBOutlet UILabel *la;
    IBOutlet UIButton *button;
    //声明对象
    UILabel *label;
    UIButton *last;
    UIButton *current;
}
- (IBAction)aa:(id)sender; //单击 Button1 按钮
- (IBAction)bb:(id)sender;
- (IBAction)cc:(id)sender;
- (IBAction)dd:(id)sender; //单击 Button4 按钮
@end
```

(7) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 aa，选中 Use Storyboard ID 复选框。

(8) 对 Aa View Controller 视图控制器的设计界面进行设计，效果如图 1.23 所示。

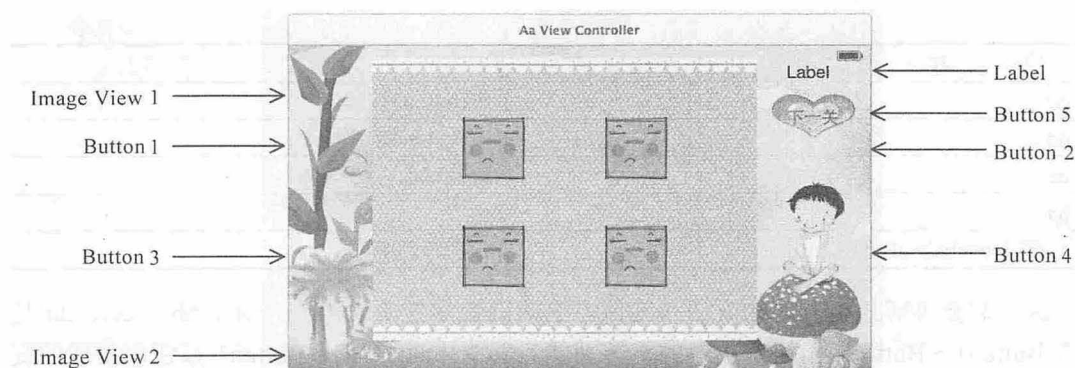


图 1.23 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设计如表 1-22 所示。

表 1-22 视图、控件设置

视图、控件	属 性 设 置	其 他
View Controller视图 控制器图标	在Simulated Metrics面板下，将Orientation属性设置为 Landscape	
Image View1	Image: 9.jpg	
Image View2	Image: 3.jpg	
Button1	Title: (空) Background: 2.jpg Tag: 1	与插座变量b1关联 与动作aa:关联
Button2	itle: (空) Background: 2.jpg Tag: 2	与插座变量b2关联 与动作bb:关联
Button3	itle: (空) Background: 2.jpg Tag: 3	与插座变量b3关联 与动作cc:关联
Button4	itle: (空) Background: 2.jpg Tag: 4	与插座变量b4关联 与动作dd:关联
Label		与插座变量label关联
Button5	Title: 下一关 Background: 1.png	与插座变量button关联

(9) 将 Play 按钮和 Aa View Controller 视图控制器中的设计界面进行关联。

(10) 打开 aaViewController.m 文件，编写代码，实现当单击两个按钮后，判断图片是否一样。使用的方法如表 1-23 所示。

表 1-23 aaViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
aa:	单击Button1按钮
bb:	单击Button2按钮

续表

方 法	功 能
cc:	单击Button3按钮
dd:	单击Button4按钮
ee	判断及赋值
pa	判断
alertView:clickedButtonAtIndex:	警告视图的响应

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。aa:、bb:、cc:、dd:是单击 Button1~Button4 按钮后实现的旋转功能。其中，aa:是单击 Button1 按钮后实现的旋转，程序代码如下：

```
- (IBAction)aa:(id)sender {
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:1];
    //设置过渡动画
    [UIView setAnimationTransition:UIViewAnimationTransitionFlipFromLeft
    forView:b1 cache:YES];
    [b1 setBackgroundImage:[UIImage imageNamed:@"5.jpg"] forState:UIControlStateNormal];
    [UIView commitAnimations];
    la.text=[NSString stringWithFormat:@"%i",b1.tag]; //设置标签的文本
    [self performSelector:@selector(ee) withObject:self afterDelay:
    0.000001];
}
```

ee 方法实现判断标签中的内容，并赋值。程序代码如下：

```
-(void)ee{
    if(current==nil){
        if([la.text isEqualToString: @"1"]){//判断标签中显示的文本是否为"1"
            current=b1;
        }else if ([la.text isEqualToString: @"2"]){
            //判断标签中显示的文本是否为"2"
            current=b2;
        }else if ([la.text isEqualToString: @"3"]){
            //判断标签中显示的文本是否为"3"
            current=b3;
        }else if ([la.text isEqualToString: @"4"]){
            //判断标签中显示的文本是否为"4"
            current=b4;
        }
    }else{
        if([la.text isEqualToString: @"1"]){//判断标签中显示的文本是否为"1"
            last=b1;
            NSLog(@"bb");
        }else if ([la.text isEqualToString: @"2"]){
            //判断标签中显示的文本是否为"2"
            last=b2;
        }else if ([la.text isEqualToString: @"3"]){
            //判断标签中显示的文本是否为"3"
            last=b3;
        }else if ([la.text isEqualToString: @"4"]){
            last=b4;
        }
    }
}
```

```

//判断标签中显示的文本是否为"4"
        last=b4;
    }
    [self performSelector:@selector(pa) withObject:self afterDelay:1];
    //经过 1 秒执行 pa 方法
}
}

```

pa 方法实现两个按钮的判断，程序代码如下：

```

-(void)pa{
if(current!=last){
    //判断 current 按钮的当前背景图像是否与 last 按钮的当前背景图像相同
    if([current.currentBackgroundImage isEqual:last.currentBackgroundImage]){
        [current setHidden:YES];
        [last setHidden:YES];
        la.text=nil;
        current=nil;
    }else{
        [current setBackgroundImage:[UIImage imageNamed:@"2.jpg"] forState:
        UIControlStateNormal];
        //设置按钮的背景图像
        [last setBackgroundImage:[UIImage imageNamed:@"2.jpg"] forState:
        UIControlStateNormal];
        la.text=nil;
        current=nil;
    }
    if([b1 isHidden]==YES&&[b2 isHidden]==YES&&[b3 isHidden]==YES&&[b4
    isHidden]==YES){
        UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"闯关成功"
        message:@"是否进入下一关" delegate:self cancelButtonTitle:@"NO"
        otherButtonTitles:@"YES", nil];
        //创建警告视图
        [alert show];
    }
}
}
}

```

alertView:clickedButtonAtIndex:方法实现警告视图的响应。程序代码如下：

```

-(void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTextAtIndex:buttonIndex];
    //判断 str 是否等于"YES"
    if([str isEqualToString:@"YES"]){
        [button setHidden:NO];
    }else{
        exit(1);
    }
}
}

```

在本实例中，翻翻看的游戏一共分为 2 个关卡，每一关卡的难度会有所不同。由于篇幅的限制，对于设计界面的设计以及代码请读者参考源代码。

【代码解析】

本实例关键功能是两个按钮的判断。下面就是这个知识点的详细讲解。

在本实例中两个按钮判断的实现，首先要声明两个按钮对象，一个用来存放当前按下的按钮值，一个用来存放最后一次按下的按钮值，代码如下：

```
UILabel *label;
UIButton *last;
```

其次，判断当前的按钮是否为空，并赋值，代码如下：

```
if(current==nil){
    if([la.text isEqualToString: @"1"]){
        current=b1;
    }
    .....
    else if ([la.text isEqualToString: @"4"]){
        current=b4;
    }
} else {
    if([la.text isEqualToString: @"1"]){
        last=b1;
    }
    .....
    else if ([la.text isEqualToString: @"4"]){
        last=b4;
    }
}
```

最后，使用 `currentBackgroundImage` 方法将当前按钮和最后一次的按钮进行判断，代码如下：

```
if(current!=last){
    if([current.currentBackgroundImage isEqual:last.currentBackgroundImage]){
        [current setHidden:YES];
        [last setHidden:YES];
        la.text=nil;
        current=nil;
    } else {
        [current setBackgroundImage:[UIImage imageNamed:@"2.jpg"] forState:
        UIControlStateNormal];
        [last setBackgroundImage:[UIImage imageNamed:@"2.jpg"] forState:
        UIControlStateNormal];
        la.text=nil;
        current=nil;
    }
    .....
}
```

实例 8 节气歌

【实例描述】

使用 `CATransition` 可以出现很多的过渡动画。本实例的功能是使用 `CATransition` 实现

立方体的动画效果，使用了节气歌的应用程序，运行效果如图 1.24 所示。

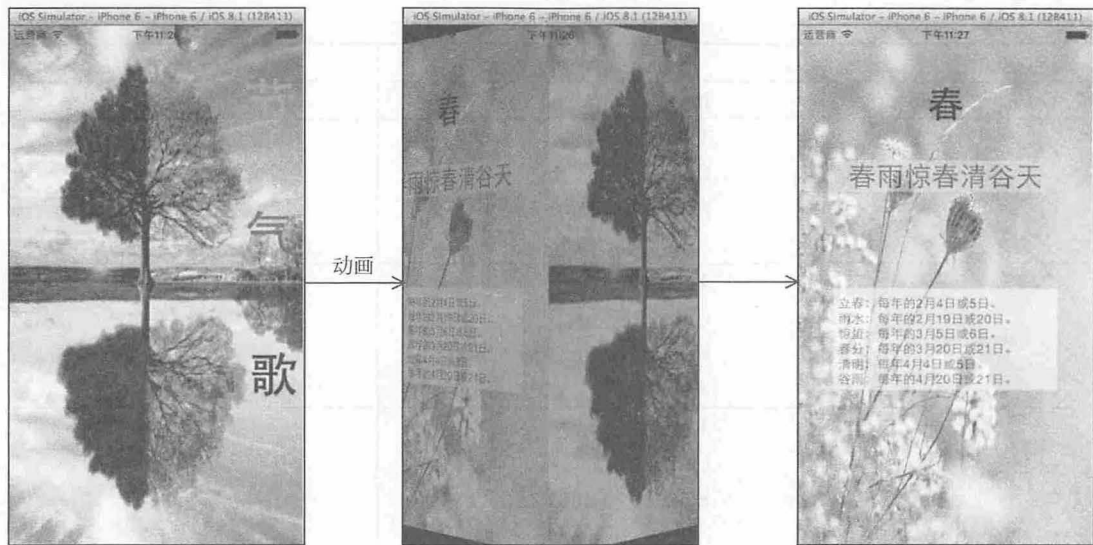


图 1.24 运行效果

【实现过程】

- (1) 创建一个项目，命名为“节气歌”。
- (2) 添加图片 1.jpg、2.jpg、3.jpg、4.jpg、5.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 单击打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.25 所示。

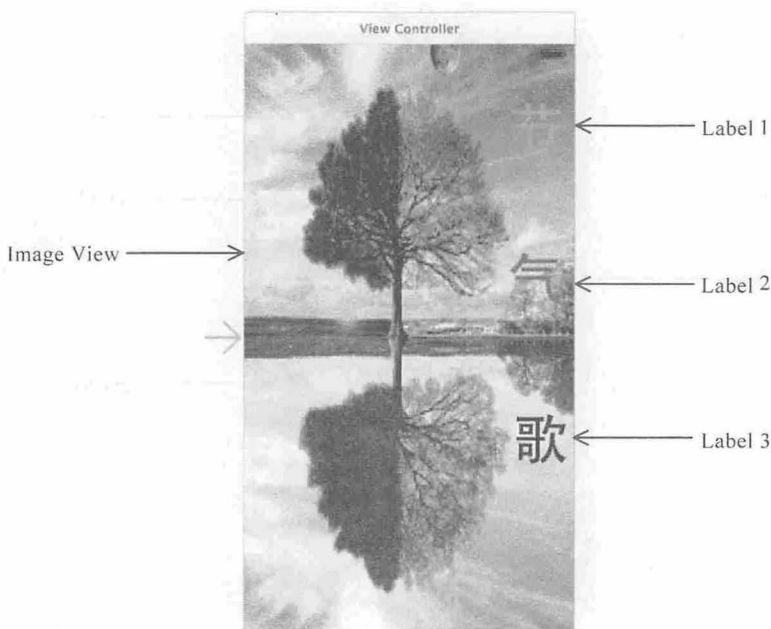


图 1.25 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-24 所示。

表 1-24 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 5.jpg	
Label1	Text: 节 Color: 绿色 Font: System Bold 53.0 Alignment: 居中	
Label2	Text: 气 Color: 红色 Font: System Bold 53.0 Alignment: 居中	
Label3	Text: 歌 Color: 黑色 Font: System Bold 53.0 Alignment: 居中	

(4) 将 Storyboard ID 设置为 vv，选中 Use Storyboard ID 复选框。

(5) 创建一个基于 UIViewController 类的 aaViewController 类。

(6) 回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 aa，选中 Use Storyboard ID 复选框。

(7) 对 Aa View Controller 视图控制器的设计界面进行设计，效果如图 1.26 所示。

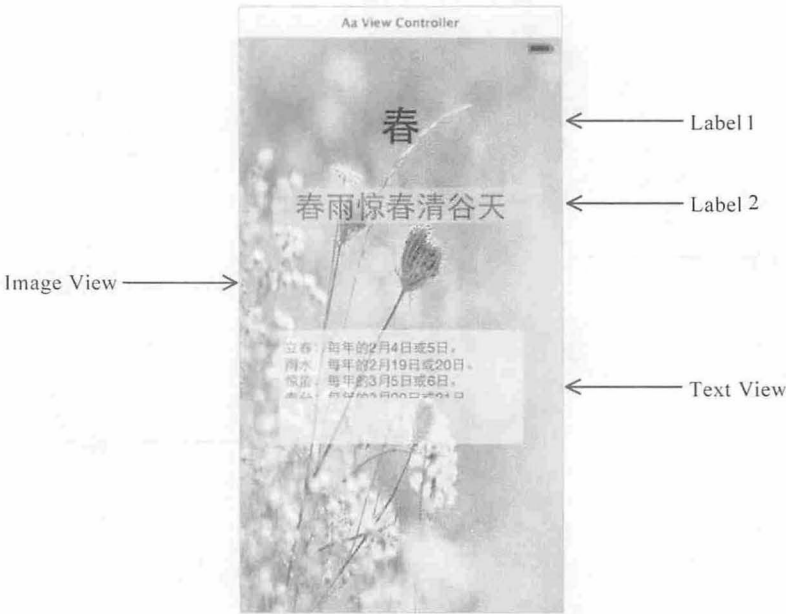


图 1.26 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-25 所示。

表 1-25 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 1.jpg	将其覆盖整个设计界面
Label1	Text: 春 Font: System Bold 40.0 Alignment: 居中	
Label2	Text: 春雨惊春清谷天 Font: System Italic 30.0 Alignment: 居中 Alpha: 0.6 Background: 绿色	
Text View	Text: 立春: 每年的2月4日或5日。 雨水: 每年的2月19日或20日。 惊蛰: 每年的3月5日或6日。 春分: 每年的3月20日或21日。 清明: 每年的4月4日或5日。 谷雨: 每年的4月20日或21日。 Editable复选框去掉选择 Alpha: 0.6	

(8) 打开 ViewController.m 文件, 编写代码, 实现视图控制器的界面切换。程序代码如下:

```

- (void)viewDidLoad
{
    [self performSelector:@selector(aa) withObject:self afterDelay:5];
    //经过 5 秒后执行 aa 方法
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

-(void)aa{
    CATransition *t=[CATransition animation];
    t.duration=2.0f;
    t.type=@"cube";
    t.subtype=kCATransitionFromLeft;
    //设置动画类型
    //设置方向
    aaViewController *aaa=[self.storyboard instantiateViewController
    WithIdentifier:@"aa"];
    [self.view addSubview:aaa.view];
    [self.view.layer addAnimation:t forKey:nil];
}

```

(9) 创建一个基于 UIViewController 类的 bbViewController 类。

(10) 回到 Main.storyboard 文件, 从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 bbViewController。这时新增的 View Controller 视图控制器就变为了 Bb View Controller 视图控制器。在 Identity 面板下, 将 Storyboard ID 设置为 bb, 选中 Use

Storyboard ID 复选框。

(11) 打开 aaViewController.m 文件，编写代码，实现视图控制器的界面切换。程序代码如下：

```
- (void) viewDidLoad
{
    [self performSelector:@selector(aa) withObject:self afterDelay:8];
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void) aa{
    CATransition *t=[CATransition animation];
    t.duration=2.0f;
    t.type=@"cube"; //设置动画类型
    t.subtype=kCATransitionFromTop; //设置方向
    bbViewController *bbb=[self.storyboard instantiateViewController
    WithIdentifier:@"bb"];
    [self.view addSubview:bbb.view];
    [self.view.layer addAnimation:t forKey:nil];
}
```

在此实例中，还有 3 个季节的界面以及代码没有给出。由于篇幅的限制，设计界面的设计以及代码请读者参考源代码。

【代码解析】

本实例关键功能是视图以正方体的形式进行切换。下面就是这个知识点的详细讲解。

在本实例中要实现立方体的动画效果，需要使用 CATransition 的 type 属性实现，代码如下：

```
t.type=@"cube";
```

对于动画出现的不同方向，可以使用 CATransition 的 subtype 方法实现。代码如下：

```
t.subtype=kCATransitionFromLeft;
```

实例 9 行走的青蛙

【实例描述】

在 UIView 动画中，有两个翻页的过渡动画，一个是 UIViewAnimationTransitionCurlUp（从下往上翻页）、一个是 UIViewAnimationTransitionCurlDown（从上往下翻页）。本实例实现的功能就是使用 UIViewAnimationTransitionCurlDown 动画效果实现青蛙的行走。运行效果如图 1.27 所示。

【实现过程】

(1) 创建一个项目，命名为“行走的青蛙”。

(2) 添加图片 1.jpg、2.jpg、3.jpg、4.jpg、5.jpg、6.jpg、7.jpg、8.jpg 到创建项目的 Supporting Files 文件夹中。

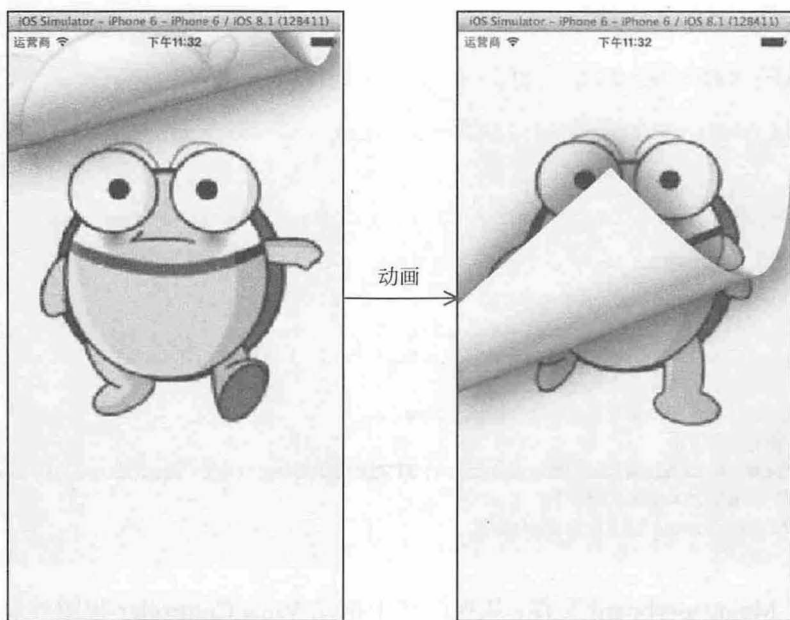


图 1.27 运行效果

(3) 单击打开 Main.storyboard 文件，在 View Controller 视图控制器的设计界面中，添加 Image View 图像视图，调整大小。单击打开 Show the Attributes inspector 图标，在 Image View 面板下，将 Image 属性设置为 1.jpg，这时设计界面效果如图 1.28 所示。



图 1.28 View Controller 视图控制器的设计界面效果

- (4) 将 Storyboard ID 设置为 vv，选中 Use Storyboard ID 复选框。
- (5) 创建一个基于 UIViewController 类的 aaViewController 类。
- (6) 打开 ViewController.m 文件，编写代码，实现视图控制器的切换效果。程序代码

如下：

```

- (void)viewDidLoad
{
    [self performSelector:@selector(aa) withObject:self afterDelay:0.5];
    //经过 0.5 秒后执行 aa 方法

    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}
- (void)aa{
    [UIView beginAnimations:@"aa" context:nil];
    [UIView setAnimationDuration:0.5];
    aaViewController *aaa=[self.storyboard instantiateViewController
    WithIdentifier:@"aa"];
    [self.view addSubview:aaa.view];
    //设置过渡动画
    [UIView setAnimationTransition:UIViewAnimationTransitionCurlDown forView:
    self.view cache:YES];
    [UIView commitAnimations];
}

```

(7) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 aa，选中 Use Storyboard ID 复选框。

(8) 在 Aa View Controller 视图控制器的设计界面中，添加 Image View 图像视图，调整大小。单击打开 Show the Attributes inspector 图标，在 Image View 面板下，将 Image 属性设置为 “2.jpg”，这时 Aa View Controller 视图控制器的设计界面效果如图 1.29 所示。

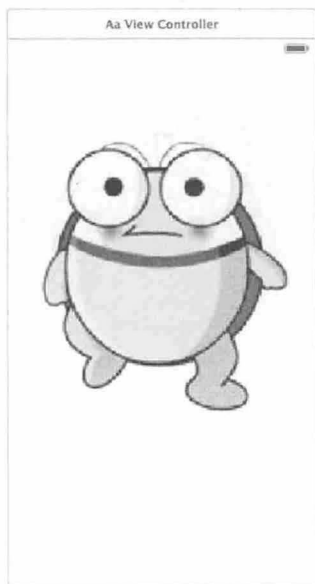


图 1.29 Aa View Controller 视图控制器的设计界面效果

(9) 创建一个基于 UIViewController 类的 bbViewController 类。

(10) 回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画

布中。

(11) 选择 Show the Identity inspector 图标，在 Custom Class 面板中，将 Class 设置为 bbViewController。这时新增的 View Controller 视图控制器就变为了 Bb View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 bb，选中 Use Storyboard ID 复选框。

(12) 在 Bb View Controller 视图控制器的设计界面中，添加 Image View 图像视图，调整大小。单击打开 Show the Attributes inspector 图标，在 Image View 面板下，将 Image 属性设置为“3.jpg”。

(13) 打开 aaViewController.m 文件，编写代码，实现视图控制器的切换效果。程序代码如下：

```
- (void)viewDidLoad
{
    [self performSelector:@selector(aa) withObject:self afterDelay:0.5];
    //经过 0.5 秒后执行 aa 方法
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}

- (void)aa{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:0.5];
    bbViewController *bbb=[self.storyboard instantiateViewController
    WithIdentifier:@"bb"];
    [self.view addSubview:bbb.view];
    //设置过渡动画
    [UIView setAnimationTransition:UIViewAnimationTransitionCurlDown forView:
    self.view cache:YES];
    [UIView commitAnimations];
}
```

在此实例中，行走动画是由 8 个动作实现的，需要类似的 8 个设计界面以及代码。由于篇幅的限制，对于设计界面的设计以及代码请读者参考源代码。

【代码解析】

本实例关键功能是动画效果的实现。下面就是这个知识点的详细讲解。

要实现动画效果，需要使用 UIView 的 setAnimationTransition:forView:cache:方法。其程序代码如下：

```
+ (void)setAnimationTransition:(UIViewAnimationTransition)transition forView:
(UIView *)view cache:
(BOOL)cache;
```

其中，(UIViewAnimationTransition)transition 表示过渡动画效果；(UIView *)view 表示过渡到的视图；(BOOL)cache 表示是否马上缓存视图内容，并且在整个过渡过程中使用缓存内容。在此代码中，就使用了 setAnimationTransition:forView:cache:方法实现了翻页动画效果，代码如下：

```
[UIView setAnimationTransition:UIViewAnimationTransitionCurlDown forView:
self.view cache:YES];
```

其中，UIViewAnimationTransitionCurlDown 表示过渡动画效果；self.view 表示过渡到的视图；YES 表示马上缓存视图内容，并且在整个过渡过程中使用缓存内容。

实例 10 变 脸

【实例描述】

CATransition 中有一个 kCATransitionReveal 的过渡动画。本实例实现的功能就是使用 kCATransitionReveal 过渡动画来实现变脸，运行效果如图 1.30 所示。

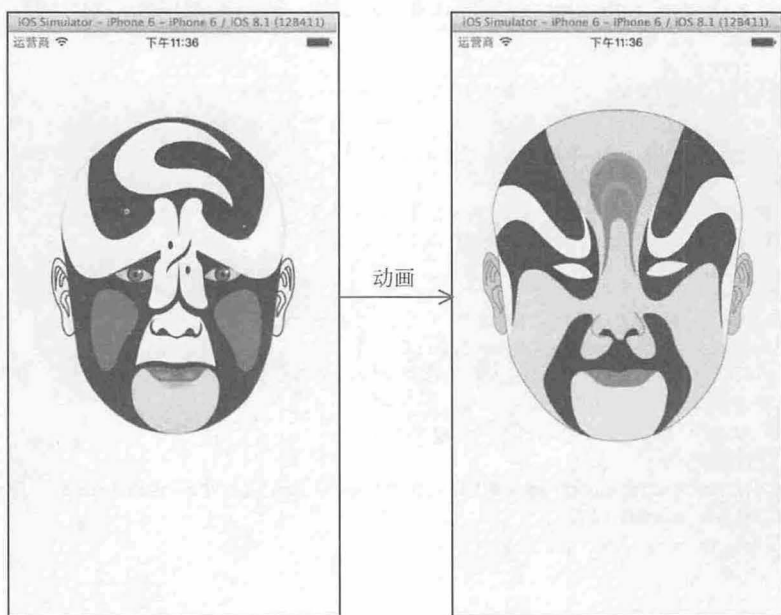


图 1.30 运行效果

【实现过程】

(1) 创建一个项目，命名为“变脸”。

(2) 添加图片 1.png、2.png、3.png、4.png、5.png 到创建项目的 Supporting Files 文件夹中。

(3) 单击打开 Main.storyboard 文件，从视图库中拖动 Image View 图像视图到设计界面，调整大小。单击 Show the Attributes inspector 图标，在 Image View 面板下，将 Image 属性设置为“li1.png”，这时设计界面效果如图 1.31 所示。

(4) 打开 ViewController.h 文件，编写代码，实现插座变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    IBOutlet UIImageView *iv1;
}
@end
```

//声明插座变量

(5) 将声明的插座变量 iv1 和设计界面的 Image View 图像视图进行关联。

(6) 打开 ViewController.m 文件，编写代码，实现变脸的效果，使用的方法如表 1-26 所示。



图 1.31 View Controller 视图控制器的设计界面效果

表 1-26 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
aa	视图脸谱的切换
aa1	
aa2	
aa3	

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，aa 方法实现脸谱 1 切换到脸谱 2 上。程序代码如下：

```
-(void)aa{
    CATransition *t=[CATransition animation];
    t.duration=0.2f;
    iv1.image=[UIImage imageNamed:@"li2.png"];
    t.type=kCATransitionReveal; //设置动画类型
    t.subtype=kCATransitionFromLeft; //设置方向
    [iv1.layer addAnimation:t forKey:nil];
    [self performSelector:@selector(aa1) withObject:self afterDelay:2];
}
```

【代码解析】

本实例关键功能是切换动画效果的实现。下面就是这个知识点的详细讲解。在本实例中切换动画效果的实现采用了 CATransition 的 type 属性，代码如下：

```
t.type=kCATransitionReveal;
```


实例 11 阴影的变化

【实例描述】

本实例的功能是为图像添加阴影，并通过滑动实现阴影的变化。运行效果如图 1.32 所示。



图 1.32 运行效果

【实现过程】

- (1) 创建一个项目，命名为“阴影的变化”。
- (2) 添加图片 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现插座变量、动作的声明，程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //声明插座变量
    IBOutlet UISlider *slider1;
    IBOutlet UISlider *slider2;
    IBOutlet UISlider *slider3;
    IBOutlet UIImageView *iv;
}
- (IBAction)aa:(id) sender;           //设置阴影的透明度
- (IBAction)bb:(id) sender;         //设置阴影的位置
- (IBAction)cc:(id) sender;         //设置阴影的扩散值
@end
```

（4）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.33 所示。

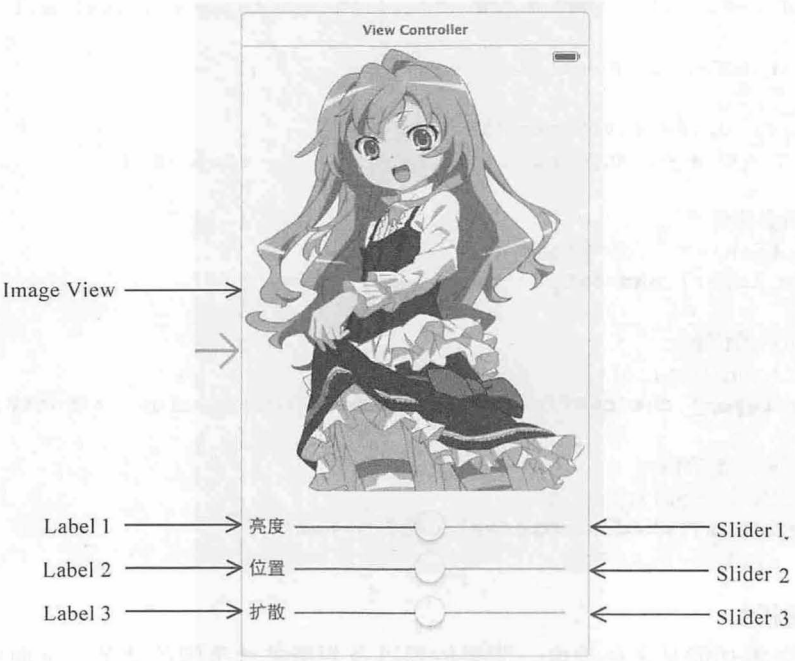


图 1.33 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-27 所示。

表 1-27 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 1.png	与插座变量iv关联
Label1	Text: 亮度 Alignment: 居中	
Slider1		与插座变量slider1关联 与动作aa:关联
Label2	Text: 位置 Alignment: 居中	
Slider2	Value的Maximum: 20 Current: 10	与插座变量slider2关联 与动作bb:关联
Label3	Title: 扩散 Alignment: 居中	
Slider3	Value的Maximum: 10 Current: 5	与插座变量slider3关联 与动作cc:关联

（5）打开 ViewController.m 文件，编写代码，实现阴影的变化。程序代码如下：

```
- (void)viewDidLoad
{
    [iv layer].shadowOffset=CGSizeMake(10, 10);           //设置阴影的位置
    [iv layer].shadowRadius=5;
```

```

    [iv layer].shadowColor=[UIColor blackColor].CGColor; //设置阴影的颜色
    [iv layer].shadowOpacity=1; //设置阴影的透明度
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}
- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}
//设置阴影的透明度
- (IBAction)aa:(id)sender {
    [iv layer].shadowOpacity=slider1.value;
}
//设置阴影的位置
- (IBAction)bb:(id)sender {
    [iv layer].shadowOffset=CGSizeMake(slider2.value, slider2.value);
}
//设置阴影的扩散值
- (IBAction)cc:(id)sender {
    [iv layer].shadowRadius=slider3.value;
}

```

【代码解析】

本实例关键功能是阴影亮度、阴影位置以及阴影扩散范围的调节。下面依次讲解这 3 个知识点。

1. 阴影亮度调节

在本实例中，阴影亮度调节使用的是 CALayer 的 shadowOpacity 属性，代码如下：

```
[iv layer].shadowOpacity=slider1.value;
```

2. 阴影位置调节

在本实例中，阴影位置调节使用的是 CALayer 的 shadowOffset 属性，代码如下：

```
[iv layer].shadowOffset=CGSizeMake(slider2.value, slider2.value);
```

3. 阴影扩散范围的调节

在本实例中，阴影扩散范围的调节使用的是 CALayer 的 shadowRadius 属性，代码如下：

```
[iv layer].shadowRadius=slider3.value;
```

实例 12 字体下载

【实例描述】

在 iOS 6 以及以后的设备中，都已安装了下载字体的功能。因此，本实例实现的功能

就是一个字体下载的功能。运行结果如图 1.34 所示。



图 1.34 运行效果

【实现过程】

- (1) 创建一个项目，命名为“下载字体”。
- (2) 添加图片 1.jpg、2.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 CoreText.framework 框架到创建的项目中。
- (4) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量、对象等的声明。

程序代码如下：

```
#import <UIKit/UIKit.h>
#import <CoreText/CoreText.h>
@interface ViewController : UIViewController{
    //声明插座变量
    IBOutlet UITextView *tv;
    IBOutlet UIProgressView *pv;
    IBOutlet UILabel *label;
    IBOutlet UILabel *la;
    //声明对象
    NSArray *fn;
    NSArray *fs;
}
@end
```

(5) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.35 所示。

需要添加的视图、控件以及对它们的设置如表 1-28 所示。

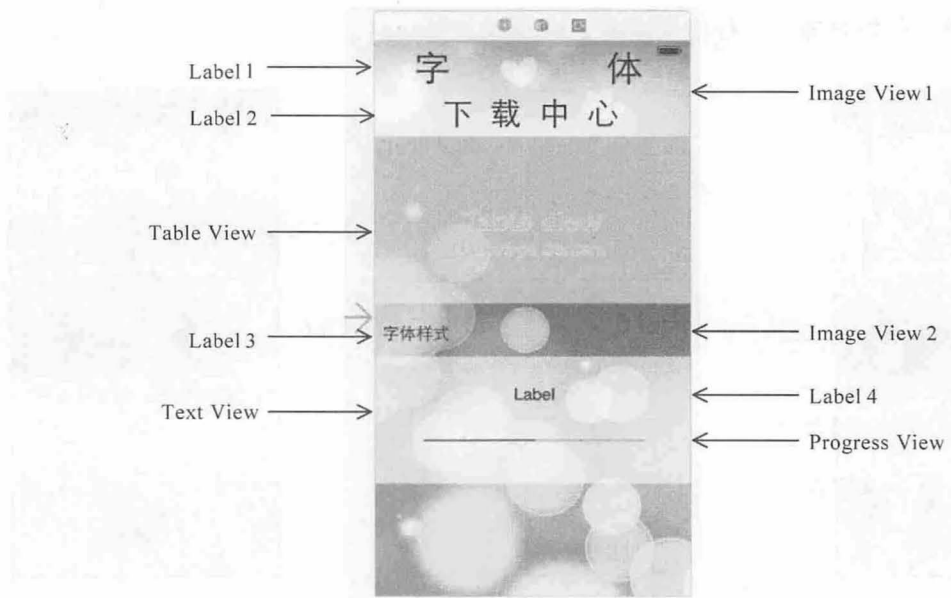


图 1.35 View Controller 视图控制器的设计界面效果

表 1-28 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View1	Image: 2.jpg	
Label1	Text: 字 体 Font: System 37.0 Alignment: 居中	
Label2	Text: 下 载 中 心 Font: System 32.0 Alignment: 居中	
Label3	Text: 字体样式 Alignment: 居中	与插座变量la关联
Label4	Alignment: 居中	与插座变量label关联
Image View2	Image: 1.jpg	
Table View		将dataSource、delegate与View Controller关联
Text View	取消选中的Editable复选框	与插座变量tv关联
Progress View		与插座变量pv关联

（6）打开 ViewController.m 文件，编写代码，实现表视图的填充以及选择字体的下载功能。使用的方法如表 1-29 所示。

表 1-29 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
tableView:numberOfRowsInSection:	获取行数
tableView:cellForRowAtIndexPath:	获取某一行的数据
asynchronouslySetFontName:	实现下载字体并显示
tableView:didSelectRowAtIndexPath:	选择行的响应

其中, `viewDidLoad`、`tableView:numberOfRowsInSection:`、`tableView:cellForRowAtIndexPath:`方法实现了表视图的填充功能。程序代码如下:

```
- (void)viewDidLoad
{
    [la setHidden:YES];
    [label setHidden:YES];
    [pv setHidden:YES];
    //创建数组 fn
    fn = [[NSArray alloc] initWithObjects:
        @"STXingkai-SC-Light",
        @"DFWaWaSC-W5",
        @"FZLTXHK--GBK1-0",
        @"STLibian-SC-Regular",
        @"LiHeiPro",
        @"HiraginoSansGB-W3",
        nil];

    //创建数组 fs
    fs = [[NSArray alloc] initWithObjects:
        @"我喜欢白天, 因为白天能做白日梦",
        @"我们陪你到老, 青春至此无悔",
        @"你的爱是个梦, 却有真实的痛",
        @"等待, 并不是期待你回来",
        @"你们的笑容, 是世界上最灿烂的阳光",
        @"在泛滥的社会中, 爱情已经不足为贵",
        nil];

    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

//获取行数
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section
{
    return [fn count];
}

//获取行的内容
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath
{
    static NSString *MyIdentifier = @"MyIdentifier";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
        MyIdentifier];
    //判断 cell 是否为空
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyle
            Default reuseIdentifier:MyIdentifier];
    }
    cell.textLabel.text = fn[indexPath.row]; //设置文本内容
    return cell;
}
```

要实现选择字体的下载, 需要使用 `asynchronouslySetFontName:`和 `tableView:didSelectRowAtIndexPath:`方法。其中, `asynchronouslySetFontName:`方法实现了对选择的字体进行下载并显示。程序代码如下:

```
- (void)asynchronouslySetFontName:(NSString *)fontName
{
    UIFont* aFont = [UIFont fontWithName:fontName size:25];
```

```

//如果字体已下载
if (aFont && ([aFont.fontName compare:fontName] == NSOrderedSame ||
[aFont.familyName compare:fontName] == NSOrderedSame)) {
    //显示示例文本
    NSUInteger sampleIndex = [fn indexOfObject:fontName];
    tv.text = [fs objectAtIndex:sampleIndex]; //设置文本
    tv.font = [UIFont fontWithName:fontName size:25]; //设置字体
    return;
}

NSMutableDictionary *attrs = [NSMutableDictionary dictionaryWithOb
jectsAndKeys:fontName, kCTFontNameAttribute, nil]; //创建字典对象
CTFontDescriptorRef desc = CTFontDescriptorCreateWithAttributes
((__bridge CFDictionaryRef)attrs);
NSMutableArray *descs = [NSMutableArray arrayWithCapacity:0];
//创建可变的数组对象

[descs addObject:(__bridge id)desc]; //添加对象
CFRelease(desc);
//字体下载
__block BOOL errorDuringDownload = NO;
CTFontDescriptorMatchFontDescriptorsWithProgressHandler( (__bridge
CFArrayRef)descs, NULL, ^(CTFontDescriptorMatchingState state,
CFDictionaryRef progressParameter) {
    double progressValue = [((__bridge NSDictionary *)progressParameter
objectForKey:(id)kCTFontDescriptorMatchingPercentage) doubleValue];
    //判断状态
    if (state == kCTFontDescriptorMatchingDidBegin) {
        dispatch_async(dispatch_get_main_queue(), ^ {
            [label setHidden:NO];
            label.text=@"正在等待下载"; //设置文本内容
            tv.text= [NSString stringWithFormat:@"Downloading %@", fontName];
            tv.font = [UIFont systemFontOfSize:14.]; //设置文本内容
        });
    } else if (state == kCTFontDescriptorMatchingDidFinish) {
        dispatch_async(dispatch_get_main_queue(), ^ {
            //显示新下载字体的文本示例
            NSUInteger sampleIndex = [fn indexOfObject:fontName];
            tv.text = [fs objectAtIndex:sampleIndex]; //设置文本内容
            tv.font = [UIFont fontWithName:fontName size:24.]; //设置字体
            CTFontRef fontRef = CTFontCreateWithName((__bridge CFString
Ref)fontName, 0., NULL);
            CFStringRef fontURL = CTFontCopyAttribute(fontRef, kCTFont
URLAttribute);
            NSLog(@"%@", (__bridge NSURL*) (fontURL)); //输出
            //释放
            CFRelease(fontURL);
            CFRelease(fontRef);
            if (!errorDuringDownload) {
                NSLog(@"%@ downloaded", fontName);
            }
        });
    }
}

//判断 state 是否为 kCTFontDescriptorMatchingWillBeginDownloading
} else if (state == kCTFontDescriptorMatchingWillBeginDownloading) {
    dispatch_async(dispatch_get_main_queue(), ^ {
        pv.progress = 0.0; //设置进度条的进度
        pv.hidden = NO;
        label.text=@"开始下载"; //设置文本内容
    });
}

```



```

        //判断 state 是否为 kCTFontDescriptorMatchingDidFinishDownloading
    } else if (state == kCTFontDescriptorMatchingDidFinishDownloading) {
        dispatch_async(dispatch_get_main_queue(), ^ {
            pv.hidden = YES;
            label.hidden=YES;                                //隐藏标签
            la.hidden=NO;
            label.text=@"下载完成";                          //设置文本内容
        });
    //判断 state 是否为 kCTFontDescriptorMatchingDownloading } else if (state ==
    kCTFontDescriptorMatchingDownloading) {
        dispatch_async(dispatch_get_main_queue(), ^ {
            [pv setProgress:progressValue / 100.0 animated:YES];
                                                    //设置进度条进度
            label.text=[NSString stringWithFormat:@"下载已完成%.0f%%",
            progressValue];
        });
    //判断 state 是否为 kCTFontDescriptorMatchingDidFailWithError
    }else if (state == kCTFontDescriptorMatchingDidFailWithError) {
        UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"下载失败"
        message:nil delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:
        nil];
                                                    //创建警告视图
        [alert show];
    };
    return (bool)YES;
    });
}

```

tableView: didSelectRowAtIndexPath:方法实现对选择行的响应,即调用 **asynchronouslySetFontName:**方法。程序代码如下:

```

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    //调用 asynchronouslySetFontName:方法实现字体的下载和显示
    [self asynchronouslySetFontName:fn[indexPath.row]];
    [self viewDidLoad];
}

```

【代码解析】

本实例关键功能是字体的下载。下面就是这个知识点的详细讲解。

在本实例中,要下载字体,首先要判断该字体是否已经下载,代码如下:

```

if (aFont && ([aFont.fontName compare:fontName] == NSOrderedSame || [aFont.
familyName compare:fontName] == NSOrderedSame)) {
    NSUInteger sampleIndex = [fn indexOfObject:fontName];
    tv.text = [fs objectAtIndex:sampleIndex];
    tv.font = [UIFont fontWithName:fontName size:25];
    return;
}

```

如果没有下载,可以先设置下载字体所需的一些参数,代码如下:

```

NSMutableDictionary *attrs = [NSMutableDictionary dictionaryWithObjectsAnd
Keys:fontName, kCTFontNameAttribute, nil];
CTFontDescriptorRef desc = CTFontDescriptorCreateWithAttributes((__bridge
CFDictionaryRef)attrs);
NSMutableDictionary *descs = [NSMutableDictionary arrayWithCapacity:0];
[descs addObject:(__bridge id)desc];

```

```
CFRelease(desc);
```

参数设置好以后，就可以实现字体的下载功能了（根据不同的状态实现不同的下载功能），代码如下：

```
block BOOL errorDuringDownload = NO;
CTFontDescriptorMatchFontDescriptorsWithProgressHandler((__bridge CFArrayRef)
descs, NULL,^(CTFontDescriptorMatchingState state, CFDictionaryRef progress
Parameter) {
double progressValue = [((__bridge NSDictionary *)progressParameter
objectForKey:(id)kCTFontDescriptorMatchingPercentage) doubleValue];
if (state == kCTFontDescriptorMatchingDidBegin) {
dispatch_async(dispatch_get_main_queue(), ^ {
[label setHidden:NO];
label.text=@"正在等待下载";
tv.text=[NSString stringWithFormat:@"Downloading %@", fontName];
tv.font=[UIFont systemFontOfSize:14.];
});
}
}
}
.....
```

实例 13 迷雾重重

【实例描述】

本实例主要实现迷雾重重的游戏。第一关，会出现多个按钮，用户单击这些按钮，但是只会随机地有一个按钮有消散迷雾的效果。第二个，随机地在某一个位置，隐藏地出现一个可以消散迷雾的按钮，用户只要按到此按钮才可以消散迷雾。第三关，在指定的时间内，用户要单击到隐藏地出现一个消散迷雾的按钮。运行效果如图 1.36 所示。

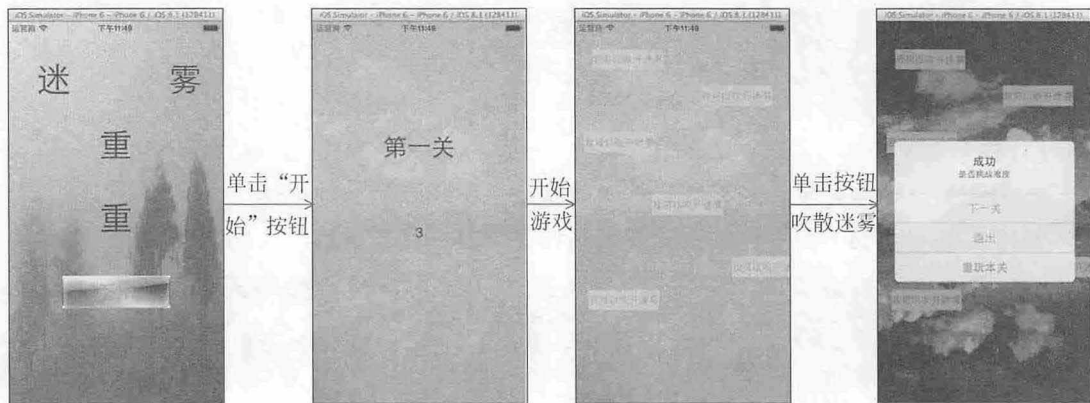


图 1.36 运行效果

【实现过程】

- (1) 创建一个项目，命名为“迷雾重重”。
- (2) 添加图片 1.jpg、2.jpg、3.jpg、4.jpg、5.jpg、6.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，

效果如图 1.37 所示。

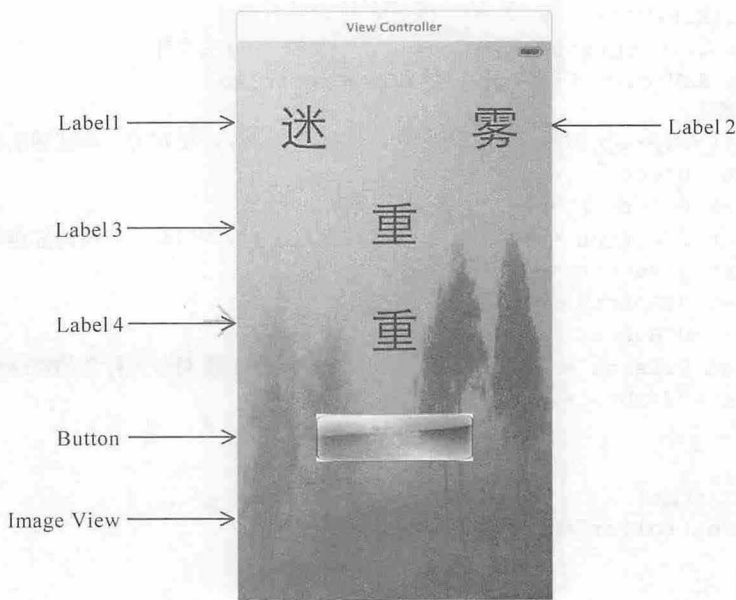


图 1.37 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-30 所示。

表 1-30 视图、控件设置

视图、控件	属性设置	其他
Image View	Image: 2.jpg	将其覆盖整个设计界面
Label1	Text: 迷 Font: System 50.0 Alignment: 居中	
Label2	Text: 雾 Font: System 50.0 Alignment: 居中	
Label3	Text: 重 Font: System 50.0 Alignment: 居中	
Label4	Text: 重 Font: System 50.0 Alignment: 居中	
Button	Title: 开始 Font: System Bold 27.0 Text Color: 绿色 Background: 1.jpg	

- (4) 创建一个基于 UIViewController 类的 aaViewController 类。
- (5) 创建一个基于 UIViewController 类的 bbViewController 类。
- (6) 打开 aaViewController.h 文件，编写代码，实现插座变量、对象、实例变量和头文

件等的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "bbViewController.h"           //头文件
@interface aaViewController : UIViewController{
    //插座变量
    IBOutlet UIImageView *iv;           //声明有关图像视图的插座变量
    IBOutlet UIButton *b1;
    IBOutlet UIButton *b2;
    IBOutlet UIButton *b3;             //声明有关按钮的插座变量
    IBOutlet UIButton *b4;
    IBOutlet UIButton *b5;
    IBOutlet UIButton *b6;
    IBOutlet UILabel *label1;          //声明有关标签的插座变量
    IBOutlet UILabel *label2;
    int j;
    //对象
    NSTimer *timer;
    bbViewController *bbb;
}
@end
```

(7) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 aa，选中 Use Storyboard ID 复选框。

(8) 对 Aa View Controller 视图控制器的设计界面进行设计，效果如图 1.38 所示。

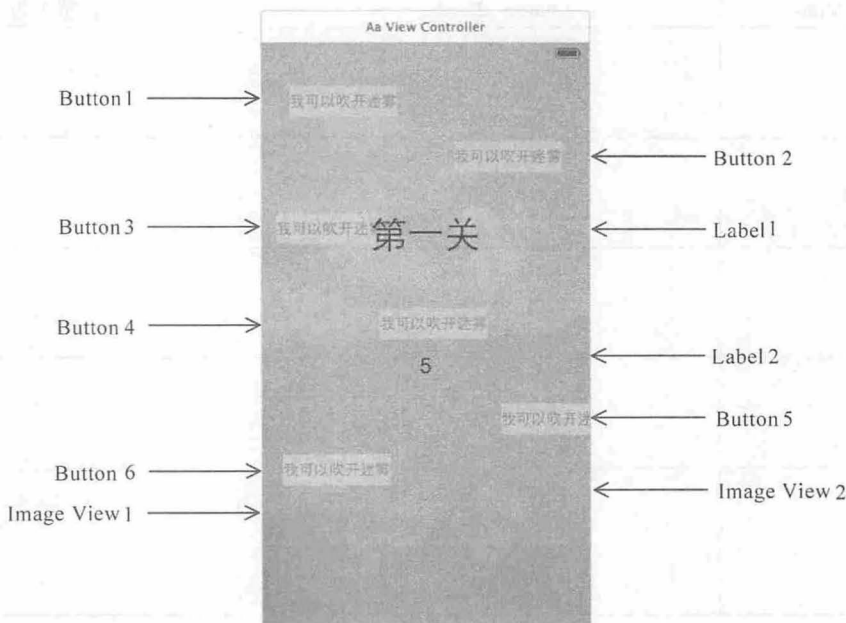


图 1.38 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-31 所示。

表 1-31 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View1	Image: 6.jpg	
Image View2	Image: 4.jpg Alpha: 0.9	放在Image View1上方 与插座变量iv关联
Label1	Text: 第一关 Font: System 36.0 Alignment: 居中	与插座变量label1关联
Label2	Text: 5 Font: System 21.0 Alignment: 居中	与插座变量label2关联
Button1	Title: 我可以吹开迷雾 Background: 青色	与插座变量b1关联
Button2	Title: 我可以吹开迷雾 Background: 青色	与插座变量b2关联
Button3	Title: 我可以吹开迷雾 Background: 青色	与插座变量b3关联
Button4	Title: 我可以吹开迷雾 Background: 青色	与插座变量b4关联
Button5	Title: 我可以吹开迷雾 Background: 青色	与插座变量b5关联
Button6	Title: 我可以吹开迷雾 Background: 青色	与插座变量b6关联

(9) 将“开始”按钮和 Aa View Controller 视图控制器的设计界面进行关联。

(10) 打开 aaViewController.m 文件，编写代码，实现迷雾的吹散、重玩本关以及进入下一关的功能，使用的方法如表 1-32 所示。

表 1-32 aaViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
time	倒计时
aa	方法的随机实现
wu:	迷雾的吹散
bb	弹出警告视图
alertView:clickedButtonAtIndex:	响应警告视图的选择

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。viewDidLoad 方法用于实现初始化功能，程序代码如下：

```
- (void)viewDidLoad
{
    //显示标签
    [label1 setHidden:NO];
    [label2 setHidden:NO];
    //隐藏按钮
    [b1 setHidden:YES];
}
```

```

[b2 setHidden:YES];
[b3 setHidden:YES];
[b4 setHidden:YES];
[b5 setHidden:YES];
[b6 setHidden:YES];
j=5;
[self performSelector:@selector(aa) withObject:self afterDelay:5];
//经过 5 秒后执行 aa 方法
timer=[NSTimer scheduledTimerWithTimeInterval:1.0 target:self selector:
@selector(time) userInfo:nil repeats:YES]; //创建时间定时器
[super viewDidLoad];
// Do any additional setup after loading the view, typically from a nib.
}

```

time 方法实现倒计时的功能。程序代码如下：

```

-(void)time{
    j--;
    label2.text=[NSString stringWithFormat:@"%i",j]; //设置标签中文本的内容
    if(j==0){
        [timer invalidate]; //让时间定时器失效
    }
}

```

aa 方法实现显示按钮，并随机地为某一按钮添加动作。程序代码如下：

```

-(void)aa{
    //隐藏标签
    [label1 setHidden:YES];
    [label2 setHidden:YES];
    //显示按钮
    [b1 setHidden:NO];
    [b2 setHidden:NO];
    [b3 setHidden:NO];
    [b4 setHidden:NO];
    [b5 setHidden:NO];
    [b6 setHidden:NO];
    //在按钮上随机出现 wu:方法
    int i=arc4random()%6;
    switch (i) {
        case 0:
            //添加动作
            [b1 addTarget:self action:@selector(wu:) forControlEvents:
            UIControlEventTouchUpInside];
            break;
        case 1:
            //添加动作
            [b2 addTarget:self action:@selector(wu:) forControlEvents:
            UIControlEventTouchUpInside];
            break;
        case 2:
            //添加动作
            [b3 addTarget:self action:@selector(wu:) forControlEvents:
            UIControlEventTouchUpInside];
            break;
        case 3:
            //添加动作
            [b4 addTarget:self action:@selector(wu:) forControlEvents:
            UIControlEventTouchUpInside];
            break;
    }
}

```

```

case 4:
    //添加动作
    [b5 addTarget:self action:@selector(wu:) forControlEvents:
    UIControlEventTouchUpInside];
default:
    //添加动作
    [b6 addTarget:self action:@selector(wu:) forControlEvents:
    UIControlEventTouchUpInside];
    break;
}
}

```

wu:方法使用渐渐消失的动画实现迷雾的吹散功能。程序代码如下：

```

- (IBAction)wu:(id)sender {
    CATransition *t=[CATransition animation];
    t.duration=5.0; //设置持续时间
    t.type=kCATransitionFade; //设置动画类型
    iv.image=[UIImage imageNamed:@"jing1.jpg"]; //设置图像视图显示的图像
    [iv.layer addAnimation:t forKey:nil];
    [self performSelector:@selector(bb) withObject:nil afterDelay:5];
}

```

alertView.clickedButtonAtIndex:方法实现对在警告视图中选择的项进行响应。程序代码如下：

```

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTextAtIndex:buttonIndex];
    //判断用户单击的按钮标题
    if([str isEqualToString:@"重玩本关"]){
        iv.image=[UIImage imageNamed:@"wu1.jpg"]; //设置图像视图显示的图像
        [self viewDidLoad];
    }else if ([str isEqualToString:@"退出"]){
        exit(1); //退出
    }else{
        bbb=[self.storyboard instantiateViewControllerWithIdentifier:@"bb"];
        [self.view addSubview:bbb.view]; //添加视图对象
    }
}

```

在此实例中，迷雾重重有3个关卡。第一个以为读者进行了详细的说明，第二个关卡是在第一关的基础上将所有的按钮进行了隐藏，由于篇幅的限制，对于第二个关卡的设计以及代码请读者参考源代码。以下将为读者详细讲解一下第三个关卡的实现，它是在第二个的基础上增加了时间的限制。

(11) 创建一个基于 UIViewController 类的 ccViewController 类。

(12) 打开 ccViewController.h 文件，编写代码，实现插座变量、实例变量、对象等的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
@interface ccViewController : UIViewController{
    //插座变量
    IBOutlet UIImageView *iv;
    IBOutlet UILabel *label1;
    IBOutlet UILabel *label2;
    IBOutlet UIButton *butt;
}

```

```
IBOutlet UILabel *label3;
IBOutlet UIView *vv;
//实例变量
CGRect ff;
int j;
int y;
//对象
NSTimer *timer;
NSTimer *ttt;
UIButton *button;
}
@end
```

(13) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 ccViewController。这时新增的 View Controller 视图控制器就变为了 Cc View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 cc，选中 Use Storyboard ID 复选框。

(14) 对 Cc View Controller 视图控制器的设计界面进行设计，效果如图 1.39 所示。

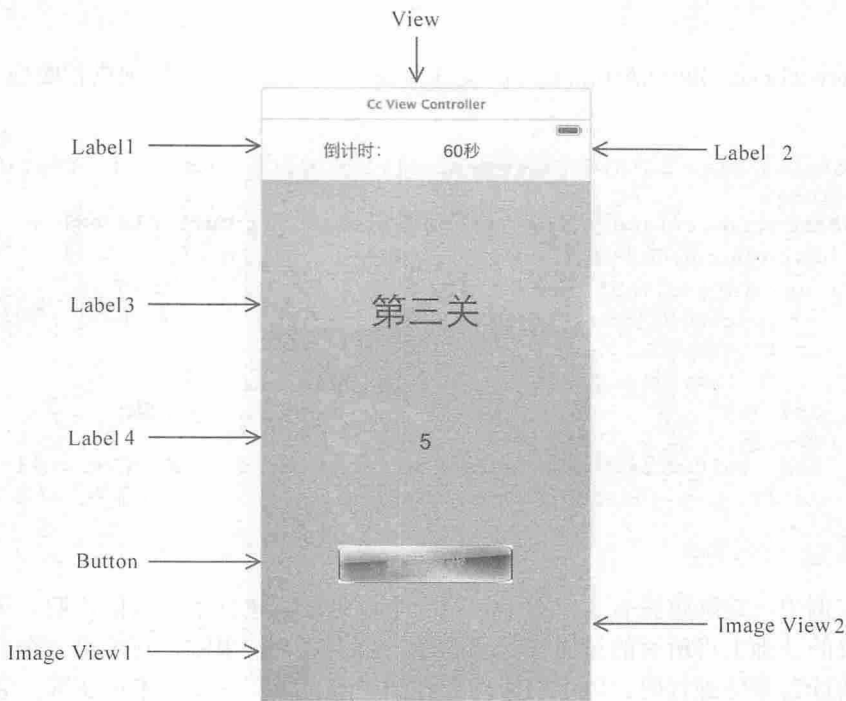


图 1.39 Cc View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-33 所示。

表 1-33 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View1	Image: 5.jpg	将其覆盖整个设计界面
Image View2	Image: 4.jpg Alpha: 0.9	在Image View1上方 与插座变量iv关联
View		与插座变量vv关联

续表

视图、控件	属性设置	其他
Label1	Text: 倒计时:	
Label2	Text: 60秒	与插座变量label3关联
Label3	Text: 第三关 Font: System 36.0 Alignment: 居中	与插座变量label11关联
Label4	Text: 5 Font: System 21.0 Alignment: 居中	与插座变量label2关联
Button	Title: 重玩游戏 Font: System Bold 20.0 Text Color: 绿色 Background: 1.jpg	与插座变量butt关联

(15) 将“重玩游戏”按钮和 View Controller 视图控制器的设计界面进行关联。

(16) 单击打开 ccViewController.m 文件, 编写代码, 实现迷雾的吹散、重玩游戏以及游戏的退出等。使用的方法如表 1-34 所示。

表 1-34 ccViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
time1	游戏开始的倒计时
aa	方法的随机实现
time2	游戏中的倒计时
wu:	迷雾的吹散
bb	弹出警告视图
alertView:clickedButtonAtIndex:	响应警告视图的选择

其中, aa 方法实现按钮的创建以及方法的随机实现。程序代码如下:

```
-(void) aa{
    [vv setHidden:NO];
    y=60;
    [self performSelector:@selector(bb) withObject:self afterDelay:60];
    //经过 60 秒后执行方法 bb
    tt=[NSTimer scheduledTimerWithTimeInterval:1.0 target:self selector:
    @selector(time2) userInfo:nil repeats:YES]; //创建时间定时器
    //隐藏标签
    [label1 setHidden:YES];
    [label2 setHidden:YES];
    int i=arc4random()%6;
    switch (i) {
        case 0:
            button=[UIButton buttonWithType:UIButtonTypeRoundedRect];
            ff=CGRectMake(212, 80, 50, 50);
            button.frame=ff; //设置按钮的框架
            [self.view addSubview:button];
            [button addTarget:self action:@selector(wu:) forControlEvents:

```

```

    UIControlEventTouchUpInside]; //添加动作
    break;
    .....
    default:
        button=[UIButton buttonWithType:UIButtonTypeRoundedRect];
        ff=CGRectMake(34, 328, 50, 50);
        button.frame=ff; //设置按钮的框架
        [self.view addSubview:button];
        [button addTarget:self action:@selector(wu:) forControlEvents:
        UIControlEventTouchUpInside]; //添加动作
        break;
    }
}

```

bb 方法实现对图像视图中的图像进行判断，从而实现警告视图的弹出。程序代码如下：

```

-(void)bb{
    if(iv.image==[UIImage imageNamed:@"5.jpg"]){
        [button removeFromSuperview];
        UIAlertView *a=[[UIAlertView alloc]initWithTitle:@"成功" message:
        @"是否退出游戏" delegate:self cancelButtonTitle:@"NO" otherButtonTitles:
        @"YES", nil]; //创建警告视图
        [a show];
    }else{
        UIAlertView *a=[[UIAlertView alloc]initWithTitle:@"失败" message:
        @"是否重玩" delegate:self cancelButtonTitle:@"重玩本关" otherButtonTitles:
        @"退出", nil];
        [a show]; //显示警告视图
    }
}

```

【代码解析】

本实例关键功能是迷雾消散的效果。下面就是这个知识点的详细讲解。

在本实例中迷雾消散的动画效果是使用了 CATransition 的公开动画 kCATransitionFade 过渡动画实现的。代码如下：

```
t.type=kCATransitionFade;
```

实例 14 重见天日

【实例描述】

本实例主要使用此过渡动画，实现重见天日的游戏。在屏幕上出现 3 个圆，用户选择一个圆作为小老鼠逃跑的路线，当遇到笼子，表示小老鼠逃跑失败；当看到户外，表示小老鼠逃跑成功。运行效果如图 1.40 所示。

【实现过程】

- (1) 创建一个项目，命名为“重见天日”。
- (2) 添加图片 1.jpg、2.jpg、3.jpg、4.jpg、5.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.41 所示。

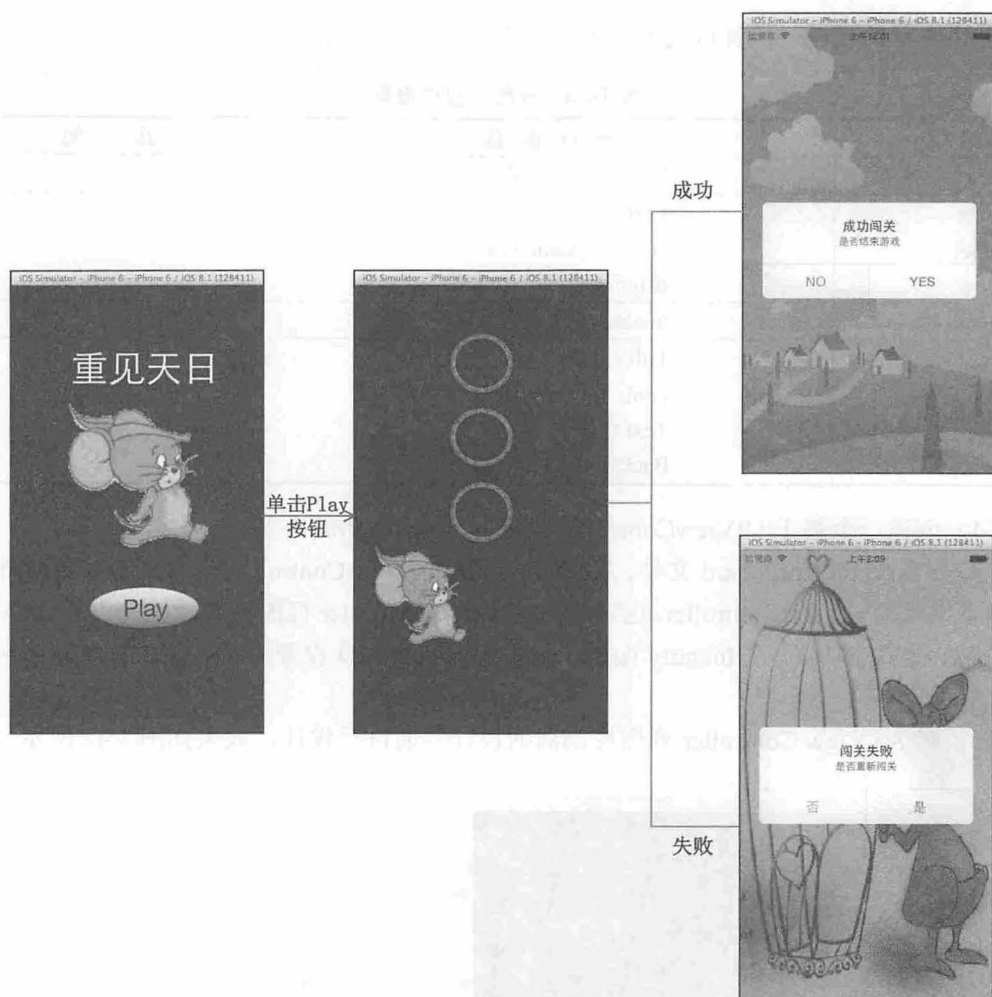


图 1.40 运行效果

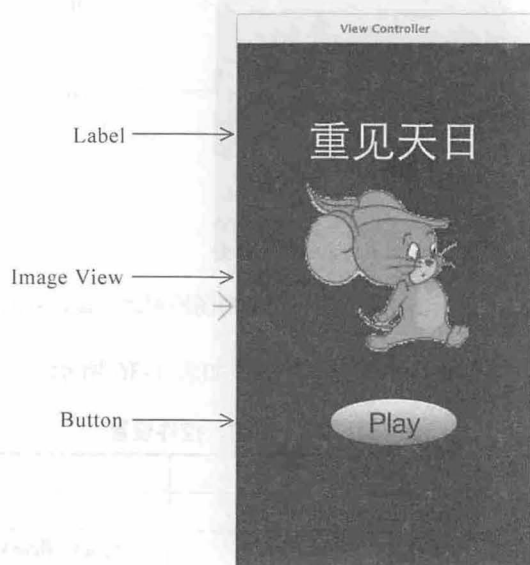


图 1.41 View Controller 视图控制器的设计界面效果

需要添加的设计、控件以及对它们的设置如表 1-35 所示。

表 1-35 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 黑色	
Label	Text: 重见天日 Font: System 47.0 Alignment: 居中对齐	
Image View	Image: 1.jpg	
Button	Title: Play Font: System 30.0 Text Color: 黑色 Background: 5.png	

(4) 创建一个基于 UIViewController 类的 aaViewController 类。

(5) 打开 Main.storyboard 文件, 从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 Identity 面板下, 将 Storyboard ID 设置为 aa, 选中 Use Storyboard ID 复选框。

(6) 对 Aa View Controller 视图控制器的设计界面进行设计, 效果如图 1.42 所示。

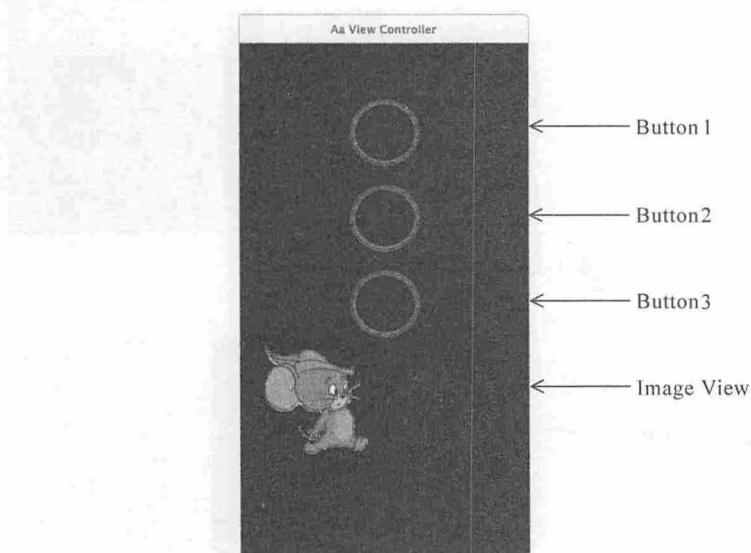


图 1.42 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-36 所示。

表 1-36 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 黑色	
Button1	Title: (空) Background: 4.jpg	在此按钮和aaViewController.h文件进行动作go1:的声明和关联

续表

视图、控件	属性设置	其他
Button2	Title: (空) Background: 4.jpg	在此按钮和aaViewController.h文件进行动作go2:的声明和关联
Button3	Title: (空) Background: 4.jpg	在此按钮和aaViewController.h文件进行动作go3:的声明和关联
Image View	Image: 1.jpg	

(7) 创建一个基于 UIViewController 类的 bbViewController 类。

(8) 打开 Main.storyboard 文件, 从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 bbViewController。这时新增的 View Controller 视图控制器就变为了 Bb View Controller 视图控制器。在 Identity 面板下, 将 Storyboard ID 设置为 bb, 选中 Use Storyboard ID 复选框。它的设计界面参考 Aa View Controller 视图控制器的设计界面效果。

(9) 打开 aaViewController.h 文件, 编写代码, 实现头文件, 以及视图控制器对象的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import "bbViewController.h"           //头文件
@interface aaViewController : UIViewController
{
    //对象
    bbViewController *bvc;
    CATransition *t;
}
- (IBAction)go1:(id)sender;           //单击 Button1 按钮实现的操作
- (IBAction)go2:(id)sender;
- (IBAction)go3:(id)sender;
@end
```

(10) 打开 aaViewController.m 文件, 编写代码, 通过镜头开的动画效果, 实现视图的切换。程序代码如下:

```
//单击 Button1 按钮实现的操作
- (IBAction)go1:(id)sender {
    t=[CATransition animation];
    t.duration=2.0f;
    t.type=@"cameraIrisHollowOpen"; //实现镜头开的动画
    int i;
    i=arc4random()%3;                //生成随机数
    switch (i) {
        case 0:
            [self.view.layer addAnimation:t forKey:nil]; //添加动画
            break;
        default:
            bvc=[self.storyboard instantiateViewControllerWithIdentifier:@"bb"];
            [self.view addSubview:bvc.view]; //添加视图对象
            [self.view.layer addAnimation:t forKey:nil];
    }
}
//单击 Button2 按钮实现的操作
- (IBAction)go2:(id)sender {
    .....
}
```



```
//单击 Button3 按钮实现的操作
- (IBAction)go3:(id)sender {
    t=[CATransition animation];
    t.duration=2.0f;                                //设置动画持续时间
    t.type=@"cameraIrisHollowOpen";
    [self.view.layer addAnimation:t forKey:nil];
}
```

(11) 创建一个基于 `UIViewController` 类的 `ccViewController` 类。

(12) 回到 `Main.storyboard` 文件，从视图库中拖动 `View Controller` 视图控制器到画布中。将 `Class` 设置为 `ccViewController`。这时新增的 `View Controller` 视图控制器就变为了 `Cc View Controller` 视图控制器。在 `Identity` 面板下，将 `Storyboard ID` 设置为 `cc`，选中 `Use Storyboard ID` 复选框。它的设计界面参考 `Aa View Controller` 视图控制器的设计界面效果。

(13) 打开 `bbViewController.h` 文件，编写代码，实现头文件，以及视图控制器对象的声明。

(14) 打开 `bbViewController.m` 文件，编写代码，通过镜头开的动画效果，实现视图的切换。至于这些代码的编写请读者参考源代码去实现。

(15) 创建一个基于 `UIViewController` 类的 `ddViewController` 类。将 `Class` 设置为 `ddViewController`。这时新增的 `View Controller` 视图控制器就变为了 `Dd View Controller` 视图控制器。在 `Identity` 面板下，将 `Storyboard ID` 设置为 `dd`，选中 `Use Storyboard ID` 复选框。

(16) 创建一个基于 `UIViewController` 类的 `eeViewController` 类。

(17) 回到 `Main.storyboard` 文件，从视图库中拖动 `View Controller` 视图控制器到画布中。将 `Class` 设置为 `eeViewController`。这时新增的 `View Controller` 视图控制器就变为了 `Ee View Controller` 视图控制器。在 `Identity` 面板下，将 `Storyboard ID` 设置为 `ee`，选中 `Use Storyboard ID` 复选框。`Dd View Controller` 视图控制器和 `Ee View Controller` 视图控制器的设计界面效果如图 1.43 所示。

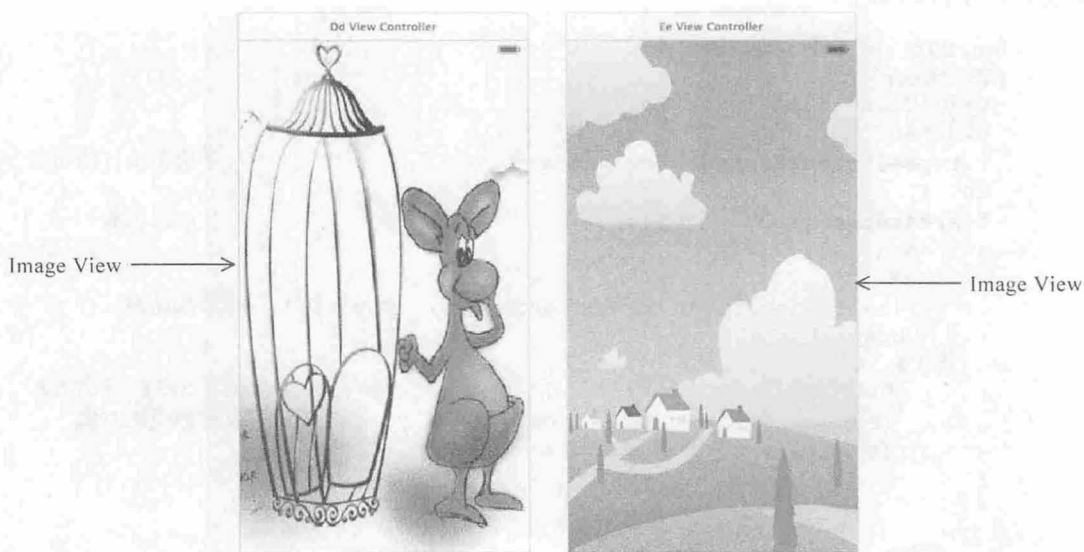


图 1.43 设置界面效果

(18) 将 Play 按钮和 Aa View Controller 视图控制器的设计界面进行关联。此时的画布效果如图 1.44 所示。

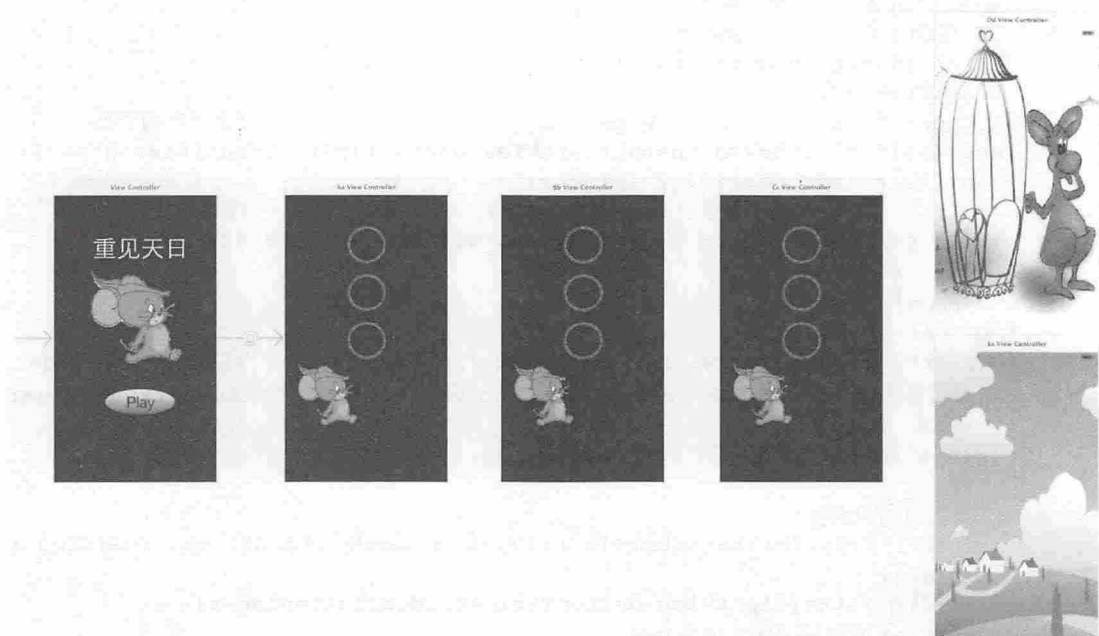


图 1.44 画布效果

(19) 打开 ccViewController.h 文件, 编写代码, 实现头文件, 以及视图控制器对象的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
//头文件
#import "eeViewController.h"
#import "ddViewController.h"
#import "ViewController.h"
@interface ccViewController : UIViewController{
    //对象
    eeViewController *evc;
    ddViewController *dvc;
    ViewController *vv;
    CATransition *t;
}
//动作
- (IBAction)go1:(id)sender;
- (IBAction)go2:(id)sender;
- (IBAction)go3:(id)sender;
@end
```

(20) 打开 ccViewController.m 文件, 编写代码, 通过镜头开的动画, 实现视图的切换, 以及警告视图的弹出和响应。程序代码如下:

```
//单击 Button1 按钮实现的操作
- (IBAction)go1:(id)sender {
    t=[CATransition animation];
    t.duration=2.0f; //设置动画持续时间
```

```

    t.type=@"cameraIrisHollowOpen";
    [self.view.layer addAnimation:t forKey:nil];
}
//单击 Button2 按钮实现的操作
- (IBAction)go2:(id)sender {
    t=[CATransition animation];
    t.duration=2.0f;
    t.type=@"cameraIrisHollowOpen"; //设置动画类型
    evc=[self.storyboard instantiateViewControllerWithIdentifier:@"ee"];
    [self.view addSubview:evc.view];
    [self.view.layer addAnimation:t forKey:nil]; //添加动画
    [self performSelector:@selector(aa) withObject:self afterDelay:3];
}
//弹出警告视图
-(void)aa{
    UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"成功闯关" message:
    @"是否结束游戏" delegate:self cancelButtonTitle:@"NO" otherButtonTitles:
    @"YES", nil]; //创建警告视图
    [alert show];
}
//响应警告视图的选择
-(void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTitleAtIndex:buttonIndex];
    //判断 str 是否等于字符串"YES"
    if([str isEqualToString:@"YES"]){
        exit(1); //退出
    }else if([str isEqualToString:@"NO"]){ //判断 str 是否等于字符串"NO"
        vv=[self.storyboard instantiateViewControllerWithIdentifier:@"vv"];
        [self.view addSubview:vv.view];
    }else if([str isEqualToString:@"是"]){ //判断 str 是否等于字符串"是"
        vv=[self.storyboard instantiateViewControllerWithIdentifier:@"vv"];
        [self.view addSubview:vv.view];
    }else if([str isEqualToString:@"否"]){ //判断 str 是否等于字符串"否"
        exit(1);
    }
}
//单击 Button3 按钮实现的操作
- (IBAction)go3:(id)sender {
    t=[CATransition animation];
    t.duration=2.0f; //设置动画持续时间
    t.type=@"cameraIrisHollowOpen";
    dvc=[self.storyboard instantiateViewControllerWithIdentifier:@"dd"];
    [self.view addSubview:dvc.view]; //添加视图对象
    [self.view.layer addAnimation:t forKey:nil];
    [self performSelector:@selector(bb) withObject:self afterDelay:3];
}
//弹出警告视图
-(void)bb{
    UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"成功失败" message:
    @"是否重新闯关" delegate:self cancelButtonTitle:@"否" otherButtonTitles:
    @"是", nil]; //创建警告视图
    [alert show];
}

```

【代码解析】

本实例关键功能是视图与视图切换时的动画效果。下面就是这个知识点的详细讲解。

在本实例中视图与视图切换时的动画效果，需要使用 `CATransition` 的非公开动画的 `@"cameraIrisHollowOpen"` 过渡动画，它实现的是镜头开的动画效果。代码如下：

```
t.type=@"cameraIrisHollowOpen";
```

实例 15 眼力测试

【实例描述】

本实例实现的功能是通过 `UIView` 自定义动画，实现眼力测试的效果。当小球进入 3 个杯子中的某一个时后，就会移动杯子。移动停止后，选择在哪个杯子有小球存在。运行效果如图 1.45 所示。

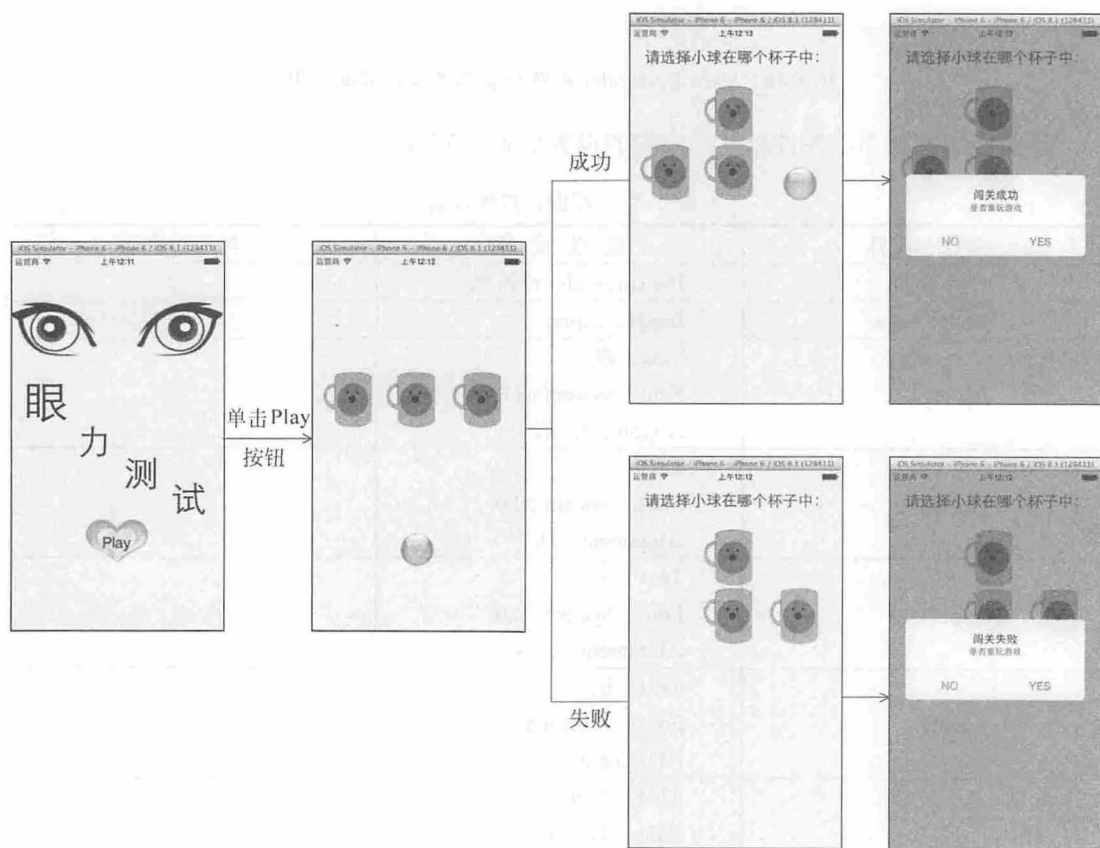


图 1.45 运行效果

【实现过程】

- (1) 创建一个项目，命名为“眼力测试”。
- (2) 添加图片 1.png、2.png、3.png、4.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.46 所示。

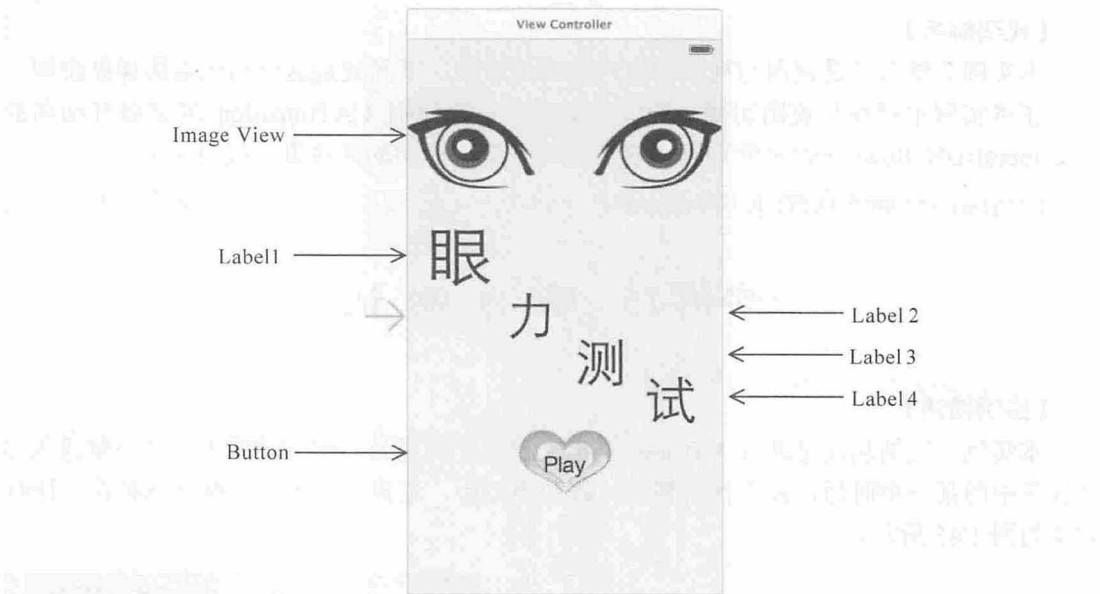


图 1.46 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-37 所示。

表 1-37 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅粉色	
Image View	Image: 2.png	
Label1	Text: 眼 Font: System 68.0 Alignment: 居中	
Label2	Text: 力 Font: System 52.0 Alignment: 居中	
Label3	Text: 测 Font: System 52.0 Alignment: 居中	
Label4	Text: 试 Font: System 52.0 Alignment: 居中	
Button	Title: Play Font: System 23.0 Text Color: 黑色 Background: 4.png	

(4) 将 Storyboard ID 设置为 vv，选中 Use Storyboard ID 复选框。

(5) 创建一个基于 UIViewController 类的 aaViewController 类。

(6) 打开 aaViewController.h 文件，编写代码，实现头文件、插座变量、对象以及动作的声明。程序代码如下：


```

#import <UIKit/UIKit.h>
#import "ViewController.h"
@interface aaViewController : UIViewController{
    //声明有关按钮的插座变量
    IBOutlet UIButton *button1;
    IBOutlet UIButton *button2;
    IBOutlet UIButton *button3;
    //声明有关图像视图的插座变量
    IBOutlet UIImageView *iv1;
    IBOutlet UIImageView *iv2;
    IBOutlet UIImageView *iv3;
    IBOutlet UIImageView *iv4;
    IBOutlet UILabel *la;
    ViewController *vcv;
}
//动作
- (IBAction)show:(id)sender;
- (IBAction)showa:(id)sender;
- (IBAction)showb:(id)sender;
@end

```

(7) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。

(8) 对 Aa View Controller 视图控制器的设计界面进行设计，效果如图 1.47 所示。

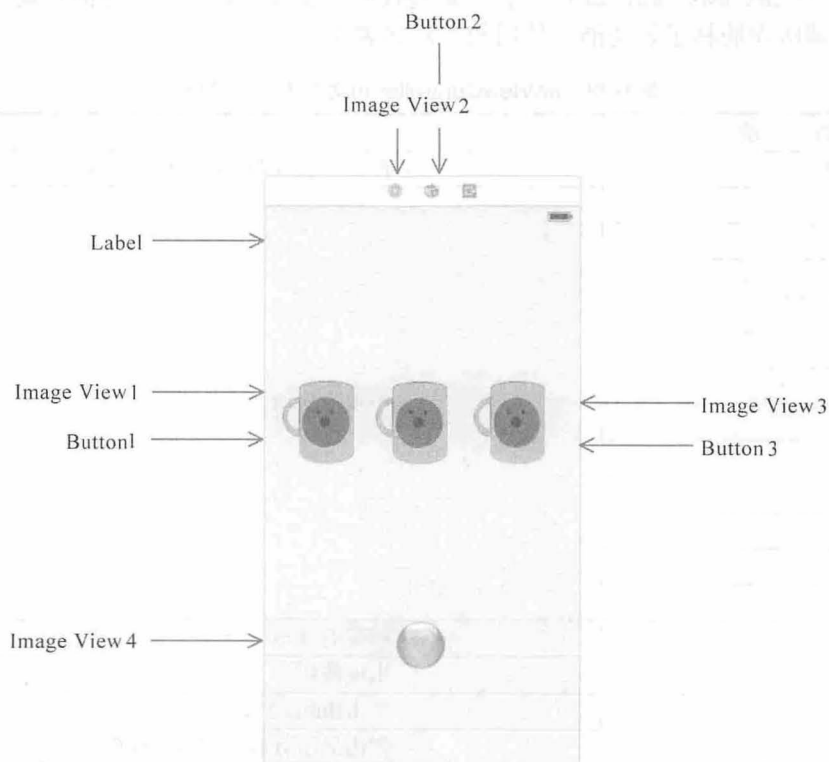


图 1.47 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-38 所示。

表 1-38 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅粉色	
Image View1	Image: 3.png	与插座变量iv1关联
Button1	Title: (空)	将其覆盖Image View1图像视图 与插座变量button1关联 与动作show:关联
Image View2	Image: 3.png	与插座变量iv2关联
Button2	Title: (空)	将其覆盖Image View2图像视图 与插座变量button2关联 与动作showa:关联
Image View3	Image: 3.png	与插座变量iv3关联
Button3	Title: (空)	将其覆盖Image View3图像视图 与插座变量button3关联 与动作showb:的声明和关联
Image View4	在Image View面板下, 将Image属性设置为“1.png”	与插座变量iv4关联
Label	在Label面板下, 将Text属性设置为“请选择小球在哪个杯子中: ” 将Font属性设置为System 23.0	将它的位置及大小设置为 (24,32,0,27)

(9) 将 Play 按钮和 Aa View Controller 视图控制器的设计界面进行关联。

(10) 打开 aaViewController.m 文件, 编写代码, 实现初始化、小球的移动、杯子的移动、选择小球所在的杯子等功能。使用的方法如表 1-39 所示。

表 1-39 aaViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
aa	小球的移动
bb1	
bb11	
bb12	
bb13	
bb2	
bb21	
bb22	
bb23	
bb3	
bb31	
bb33	
cc	标签的显示
caa	出现按钮
show:	单击Button1按钮
dd	弹出显示有成功的警告视图
alertView:clickedButtonAtIndex:	警告视图选择的响应
ee	弹出显示有失败的警告视图
showa:	单击Button2按钮
showb:	单击Button3按钮

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中,初始化的实现需要使用 `viewDidLoad` 方法实现。程序代码如下:

```
-(void)viewDidLoad
{
    //隐藏按钮
    [button1 setHidden:YES];
    [button2 setHidden:YES];
    [button3 setHidden:YES];
    //设置按钮的 tag 值
    button1.tag=0;
    button2.tag=0;
    button3.tag=0;
    [self performSelector:@selector(aa) withObject:self afterDelay:3];
    //经过 3 秒后执行 aa 方法
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}
```

小球的移动需要使用 `aa` 方法来实现。程序代码如下:

```
-(void)aa{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2];
    int i=arc4random()%3;
    switch (i) {
        case 0:
            iv4.frame=CGRectMake(25, 197, 80, 80);
            [UIView commitAnimations];
            button1.tag=2;
            [self performSelector:@selector(bb1) withObject:self afterDelay:3.5];
            break;
        case 1:
            iv4.frame=CGRectMake(119, 197, 80, 80);
            [UIView commitAnimations];
            button2.tag=2;
            [self performSelector:@selector(bb2) withObject:self afterDelay:3.5];
            break;
        default:
            iv4.frame=CGRectMake(218, 197, 80, 80);
            [UIView commitAnimations];
            button3.tag=2;
            [self performSelector:@selector(bb3) withObject:self afterDelay:3.5];
            break;
    }
}
```

杯子发生了两次移动,其中第一次移动使用了 `bb1`、`bb2`、`bb3` 方法。`bb1` 实现的是当小球进入到 `Image View1` 后,实现的第一次移动。程序代码如下:

```
-(void)bb1{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2];
    int i=arc4random()%3;
    switch (i) {
        case 0:
            //设置框架
            iv1.frame=CGRectMake(113, 178, 91, 89);
```

```

        iv2.frame=CGRectMake(18, 178, 91, 89);
        iv4.frame=CGRectMake(119, 197, 80, 80);
        button1.frame=CGRectMake(113, 178, 91, 89);
        button2.frame=CGRectMake(18, 178, 91, 89);
        [UIView commitAnimations];
        [self performSelector:@selector(bb11) withObject:self afterDelay:3];
        //经过 3 秒后执行 bb11 方法

        break;
    case 1:
        //设置框架
        iv1.frame=CGRectMake(212, 178, 91, 89);
        iv3.frame=CGRectMake(18, 178, 91, 89);
        iv4.frame=CGRectMake(218, 197, 80, 80);
        button1.frame=CGRectMake(212, 178, 91, 89);
        button3.frame=CGRectMake(18, 178, 91, 89);
        [UIView commitAnimations];
        [self performSelector:@selector(bb12) withObject:self afterDelay:3];
        //经过 3 秒后执行 bb12 方法

        break;
    default:
        //设置框架
        iv2.frame=CGRectMake(212, 178, 91, 89);
        iv3.frame=CGRectMake(113, 178, 91, 89);
        button2.frame=CGRectMake(212, 178, 91, 89);
        button3.frame=CGRectMake(113, 178, 91, 89);
        [UIView commitAnimations];
        [self performSelector:@selector(bb13) withObject:self afterDelay:3];
        //经过 3 秒后执行 bb13 方法

        break;
    }
}

```

第二次移动使用的方法有 bb11、bb12、bb13、bb21、bb22、bb23、bb31、bb33。其中，bb11 方法是当小球进入到 Image View1 后，实现的第二次移动。程序代码如下：

```

-(void)bb11{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2];
    //设置框架
    iv1.frame=CGRectMake(212, 178, 91, 89);
    iv3.frame=CGRectMake(113, 178, 91, 89);
    iv4.frame=CGRectMake(218, 197, 80, 80);
    button1.frame=CGRectMake(212, 178, 91, 89);
    button3.frame=CGRectMake(113, 178, 91, 89);
    [UIView commitAnimations];
    [self performSelector:@selector(cc) withObject:self afterDelay:3];
    //经过 3 秒后执行 cc 方法
}

```

要实现选择小球所在的杯子，需要使用 cc、caa、show:、dd、alertView:clickedButtonAtIndex:、showa:、showb:方法。其中，cc 方法实现标签以动画的形式显示。程序代码如下：

```

-(void)cc{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2];
    la.frame=CGRectMake(24, 32, 279, 27); //设置框架
    [UIView commitAnimations];
    [self performSelector:@selector(caa) withObject:self afterDelay:2.5];
    //经过 2.5 秒后执行 caa 方法
}

```

show:方法实现单击 Button1 按钮后, 移动 Image View 以及判断。程序代码如下:

```

- (IBAction)show:(id) sender {
    if(button1.tag==2){
        [UIView beginAnimations:@" " context:nil];
        [UIView setAnimationDuration:2];           //设置动画持续时间
        iv1.frame=CGRectMake(113, 90, 91, 89);
        [UIView commitAnimations];
        [self performSelector:@selector(dd) withObject:self afterDelay:3];
                                                //经过 3 秒后执行 dd 方法
    }else{
        [UIView beginAnimations:@" " context:nil];
        [UIView setAnimationDuration:2];
        iv1.frame=CGRectMake(113, 90, 91, 89);     //设置框架
        [UIView commitAnimations];
        [self performSelector:@selector(ee) withObject:self afterDelay:3];
                                                //经过 3 秒后执行 ee 方法
    }
}

```

dd、alertView:clickedButtonAtIndex:方法是弹出警告视图, 并对警告视图的选择进行响应。程序代码如下:

```

//弹出应该视图
-(void)dd{
    UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"闯关成功" message:
@"是否重玩游戏" delegate:self cancelButtonTitle:@"NO" otherButtonTitles:
@"YES", nil] ;
    [alert show];
}
//响应警告视图
-(void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTitleAtIndex:buttonIndex];
    //判断 str 是否等于"YES"
    if ([str isEqualToString:@"YES"]) {
        vcv=[self.storyboard instantiateViewControllerWithIdentifier:@"vv"];
        [self.view addSubview:vcv.view];           //添加视图对象
    }else {
        exit(1);
    }
}

```

【代码解析】

本实例关键功能是小球和杯子的移动以及杯子的选择。下面依次讲解这两个知识点。

1. 小球和杯子的移动

在本实例中小球和杯子的移动实现的方法都是一样的, 它们都使用了块动画。以小球的某一移动为例, 代码如下:

```

[UIView beginAnimations:@" " context:nil];
[UIView setAnimationDuration:2];
int i=arc4random()%3;
switch (i) {
    case 0:
        iv4.frame=CGRectMake(25, 197, 80, 80);

```



```

[UIView commitAnimations];
button1.tag=2;
[self performSelector:@selector(bb1) withObject:self afterDelay:3.5];
break;
.....
}
}
}

```

2. 杯子的选择

在本实例中杯子的选择，使用的方法是对 tag 值进行判断。首先，根据小球移动停止的地方，为在此地方的按钮设置 tag 值，设置为 2。然后，当标签出现后，实现选择功能，当单击某一按钮，就对 tag 值进行判断，是否为设置的 tag 值。以单击 Button1 为例，代码如下：

```

- (IBAction)show:(id)sender {
    if(button1.tag==2){
        //tag 值为 2
        .....
    }else{
        //tag 值不为 2
        .....
    }
}
}

```

实例 16 变化的方阵

【实例描述】

本实例的功能是通过单击屏幕，出现很多由小变大的方阵。运行效果如图 1.48 所示。

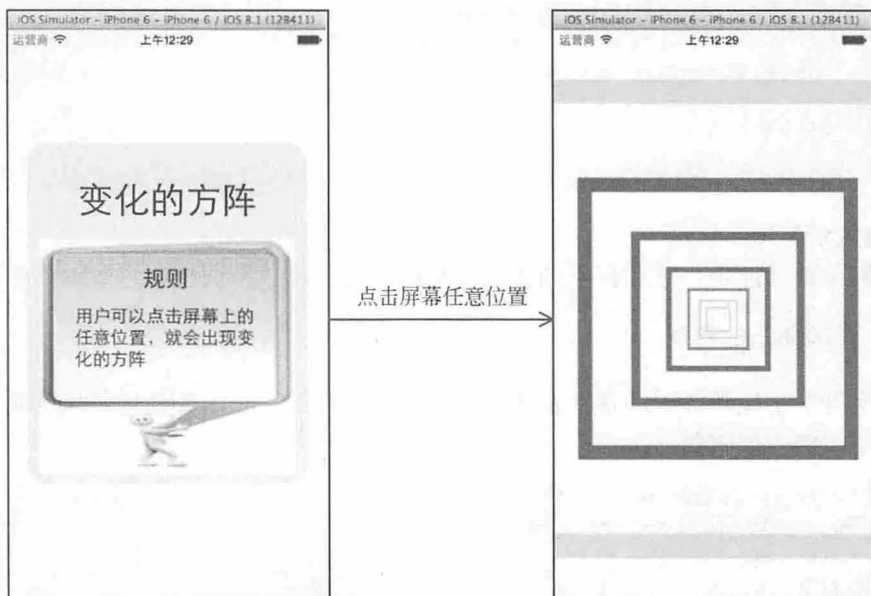


图 1.48 运行效果

【实现过程】

- (1) 创建一个项目，命名为“变化的方阵”。
- (2) 添加图片 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现插座变量、对象以及动作的声明。

程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    CALayer *layer;
    //插座变量
    IBOutlet UIView *vv;
    IBOutlet UIButton *button;
}
- (IBAction)change:(id)sender;
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.49 所示。



图 1.49 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-40 所示。

表 1-40 视图、控件设置

视图、控件	属 性 设 置	其 他
Button		将其覆盖整个设计界面 与动作change:关联 与插座变量button关联
View	Background: 浅粉色	与插座变量vv关联
Label1	Text: 变化的方阵 Font: System 36.0 Alignment: 居中	

续表

视图、控件	属 性 设 置	其 他
Image View	Image: 1.jpg	
Label2	Text: 规则 Font: System 22.0 Alignment: 居中	
Text View	Text: 用户可以点击屏幕上的任意位置, 就会出现变化的方阵 Font: System 18.0 Behavior的Editable复选框取消 Background: Default	

(5) 打开 ViewController.m 文件, 编写代码, 实现方阵的变化动画。程序代码如下:

```
//视图加载后调用, 实现初始化
- (void)viewDidLoad
{
    [button setHidden:YES];
    vv.layer.cornerRadius=20.0f; //设置圆角半径
    [self performSelector:@selector(aa) withObject:self afterDelay:3]; //经过 3 秒后执行 aa 方法
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}
//实现提示窗口的移动即 View 的移动
- (void)aa{
    [UIView beginAnimations:@" " context:nil];
    [UIView setAnimationDuration:2]; //设置动画持续时间
    vv.frame=CGRectMake(20, -343, 280, 342); //设置框架
    [UIView commitAnimations];
    [button setHidden:NO]; //隐藏按钮
}
//实现方阵颜色的改变
- (IBAction)change:(id)sender {
    layer=[CALayer layer];
    layer.frame=CGRectMake(160, 284, 10, 10); //设置框架
    int color=arc4random()%7;
    switch (color) {
        case 0:
            layer.borderColor=[UIColor yellowColor].CGColor; //设置边框颜色
            break;
        case 1:
            layer.borderColor=[UIColor redColor].CGColor;
            break;
        case 2:
            layer.borderColor=[UIColor blackColor].CGColor; //设置边框颜色
            break;
        case 3:
            layer.borderColor=[UIColor grayColor].CGColor;
            break;
        case 4:
            layer.borderColor=[UIColor brownColor].CGColor; //设置边框颜色
            break;
        case 5:
            layer.borderColor=[UIColor blueColor].CGColor;
```

```

        break;
    case 6:
        layer.borderColor=[UIColor cyanColor].CGColor; //设置边框颜色
        break;
    default:
        layer.borderColor=[UIColor greenColor].CGColor;

        break;
}
layer.borderWidth=0.5; //设置边框宽度
[self.view.layer addSublayer:layer];
[self scale:layer];
}
//实现缩放
-(void)scale:(CALayer *)alayer{
    float max=120.0;
    if(alayer.transform.m11<max){
        if(alayer.transform.m11==1.0){
            alayer.transform=CATransform3DMakeScale(1.1, 1.1, 1.0); //缩放
        }else{
            alayer.transform=CATransform3DScale(alayer.transform, 1.1, 1.1, 1.0); //缩放
        }
        [self performSelector:_cmd withObject:alayer afterDelay:0.1];
        //经过 0.1 秒后执行 _cmd 方法
    }else{
        [alayer removeFromSuperlayer]; //移除图层
    }
}
}

```

【代码解析】

本实例关键功能是方阵的改变。下面就是这个知识点的详细讲解。

要实现方阵的改变，需要使用 CALayer 的 transform 属性，其语法形式如下：

```
@property CATransform3D transform;
```

在此代码中，就是使用 transform 属性实现了方阵的缩放功能。代码如下：

```

if(alayer.transform.m11==1.0){
    alayer.transform=CATransform3DMakeScale(1.1, 1.1, 1.0);
}else{
    alayer.transform=CATransform3DScale(alayer.transform, 1.1, 1.1, 1.0);
}

```

实例 17 调 色 板

【实例描述】

setBackgroundColor:用来对视图的颜色进行设置，它不仅设计单种颜色，还可以设置 RGB 等。本实例实现的功能是通过 setBackgroundColor:来制作一个调色板。当用户出现 RGB 的数值后，单击调色按钮，就可以出现调色的结果。运行效果如图 1.50 所示。

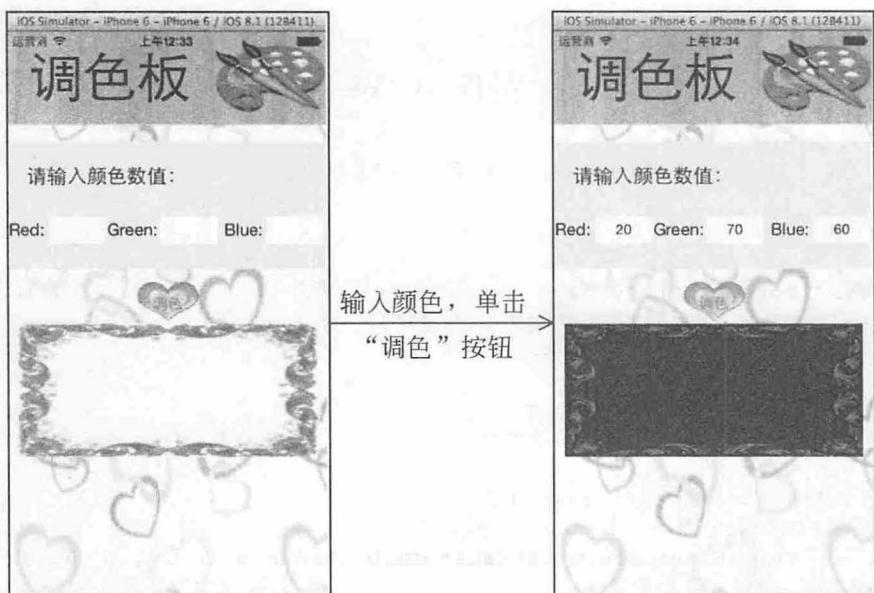


图 1.50 运行结果

【实现过程】

- (1) 创建一个项目，命名为“调色板”。
- (2) 添加图片 1.jpg、2.jpg、3.png、4.png、5.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现插座变量、实例变量、对象、动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //声明有关文本框的插座变量
    IBOutlet UITextField *t1;
    IBOutlet UITextField *t2;
    IBOutlet UITextField *t3;
    //声明有关 NSString 的对象
    NSString *text1;
    NSString *text2;
    NSString *text3;
    //声明有关 CGFloat 的对象
    CGFloat aa;
    CGFloat bb;
    CGFloat cc;
    IBOutlet UIView *vv;           //声明有关视图的插座变量
}
- (IBAction)aa:(id)sender;
- (IBAction)bb:(id)sender;
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.51 所示。

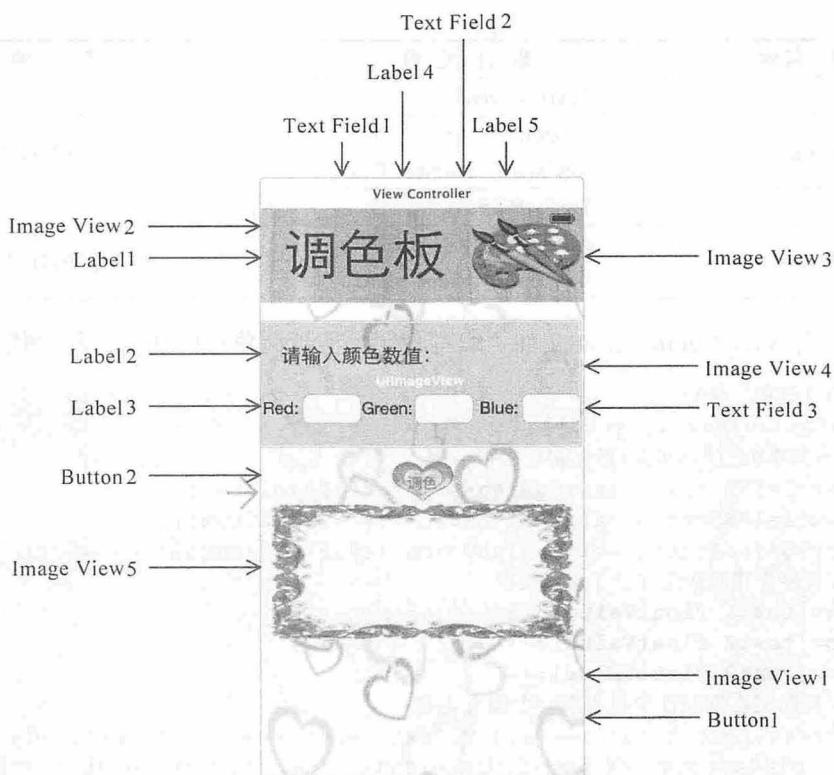


图 1.51 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-41 所示。

表 1-41 视图、控件设置

视图、控件	属性设置	其他
Button1	Title: (空)	覆盖整个设计界面 与动作bb:关联
Image View1	Image: 2.jpg	
Image View2	Image: 1.jpg	
Image View3	Image: 3.png	
Image View4	Background: 浅青色	
Image View5	Image: 4.png	
Label1	Text: 调色板 Font: System 54.0 Alignment: 居中	
Button2	Title: 调色 Background: 5.png	与动作aa:关联
Label2	Text: 请输入颜色数值: Font: System 20.0	
Label3	Text: Red:	
Text Field1	Alignment: 居中对齐 Keyboard: Number Pad	与插座变量t1关联

续表

视图、控件	属 性 设 置	其 他
Label4	Text: Green:	
Text Field2	Alignment: 居中对齐 Keyboard: Number Pad	与插座变量t2关联
Label5	Text: Blue:	
Text Field3	Alignment: 居中对齐 Keyboard: Number Pad	与插座变量t3关联

(5) 打开 ViewController.m 文件，编写代码，实现调色板的功能。程序代码如下：

```
//单击“调色”按钮
- (IBAction)aa:(id)sender {
    //将文本框控件中的内容保存在字符串中
    text1=[NSString stringWithFormat:@"%d",t1.text];
    text2=[NSString stringWithFormat:@"%d",t2.text];
    text3=[NSString stringWithFormat:@"%d",t3.text];
    //将字符串转换为 CGFloat 类型
    aa=[text1 floatValue];
    bb=[text2 floatValue];
    cc=[text3 floatValue];
    //判断文本框控件中是否有一个没有内容
    if((t1.text.length==0||(t2.text.length==0||(t3.text.length==0){
        UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"提示"
        message:@"3 个文本框不能为空" delegate:self cancelButtonTitle:@"知道啦"
        otherButtonTitles:nil]; //创建警告视图
        [alert show];
    }else{
        //判断用户输入的值是否在 0~255 之间
        if((aa>=0.0&&aa<=255) && (bb>=0.0&&bb<=255) && (cc>=0.0&&cc<=255)){
            [self performSelector:@selector(cc) withObject:self afterDelay:
            0.2]; //经过 0.2 秒后执行 cc 方法
        }else{
            UIAlertView *action=[[UIAlertView alloc] initWithTitle:@"提示"
            message:@"请不要超出 0.0~255 这个范围" delegate:nil cancelButtonTitle:
            Title:@"知道啦" otherButtonTitles: nil];
            [action show];
        }
    }
}

//实现调色功能
-(void)cc{
    //设置背景颜色
    [vv setBackgroundColor:[UIColor colorWithRed:aa/255 green:bb/255
    blue:cc/255 alpha:1.0]];
}

//关闭键盘
- (IBAction)bb:(id)sender {
    [t1 resignFirstResponder];
    [t2 resignFirstResponder];
    [t3 resignFirstResponder];
}
```

【代码解析】

本实例关键功能是在输入颜色值后，会在指定的视图中出现相应的背景颜色。下面就是这个知识点的详细讲解。

如果想要实现在输入颜色值后，在指定的视图中出现相应的背景色，首先，需要使用 floatValue 方法将字符串转换为浮点类型，代码如下：

```
aa=[text1 floatValue];
bb=[text2 floatValue];
cc=[text3 floatValue];
```

然后，使用 setBackgroundColor:方法设置指定视图的背景颜色，代码如下：

```
[vv setBackgroundColor:[UIColor colorWithRed:aa/255 green:bb/255 blue:cc/255 alpha:1.0]];
```

实例 18 量 尺

【实例描述】

本实例实现的功能是使用 Quartz2D 二维图形绘制引擎来实现量尺的绘制。运行效果如图 1.52 所示。

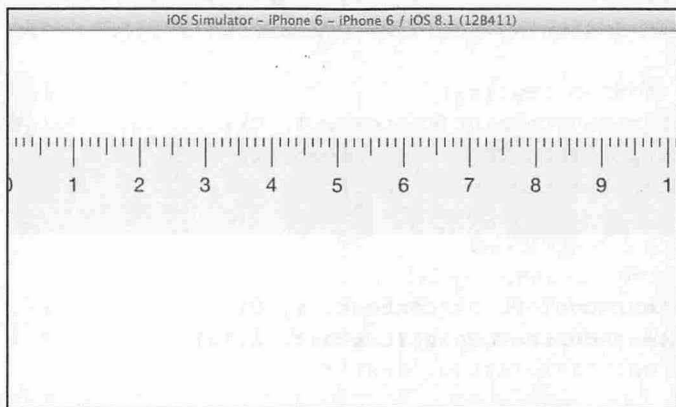


图 1.52 运行效果

【实现过程】

- (1) 创建一个项目，命名为“量尺”。
- (2) 打开目标窗口，选择 General 选项，在 Deployment Info 面板下，只选中 Landscape Left 复选框和 Landscape Right 复选框。
- (3) 打开 Main.storyboard 文件，单击 View Controller 视图控制器的 dock 工作区的 View Controller 图标，单击 Show the Attributes 后，在 Simulate Metrics 面板下，将 Orientation 属性设置为 Landscape。
- (4) 在 View Controller 视图控制器的设计界面中，拖放一个 View 空白视图，将 Background 属性设置为浅绿色。这时设置界面的效果如图 1.53 所示。

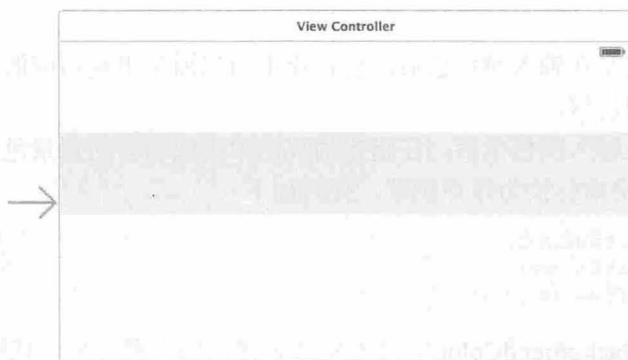


图 1.53 View Controller 视图控制器的设计界面效果

(5) 创建一个基于 UIView 类的 aa 类。

(6) 打开 aa.m 文件，编写代码，实现刻度的显示。程序代码如下：

```

- (void)drawRect:(CGRect)rect
{
    CGContextRef context=UIGraphicsGetCurrentContext(); //获取图形上下文
    //绘制间隔为 1 厘米的线段
    for (int i=0,j=0; i<568; i+=56) {
        UILabel *la=[[UILabel alloc] initWithFrame:CGRectMake(i-5, 30, 10,
            20)];
        la.text=[NSString stringWithFormat:@"%i",j]; //设置文本内容
        j++;
        [self addSubview:la]; //添加视图对象
        CGContextMoveToPoint(context, i, 0); //设置起点
        CGContextAddLineToPoint(context, i, 25); //设置终点
        CGContextStrokePath(context); //绘制路径
    }
    //绘制间隔为 0.5 厘米的线段
    for (int i=0; i<568; i+=28) {
        CGContextMoveToPoint(context, i, 0); //设置起点
        CGContextAddLineToPoint(context, i,15); //设置终点
        CGContextStrokePath(context);
    }
    //绘制间隔为 0.1 厘米的线段
    for (int i=0; i<568; i+=7) {
        CGContextMoveToPoint(context, i, 0); //设置起点
        CGContextAddLineToPoint(context, i,8); //设置终点
        CGContextStrokePath(context);
    }
}

```

(7) 单击打开 Main.storyboard 文件，再单击设计界面中的浅绿色空白视图，然后单击 Show the Identity inspector 图标，在 Custom Class 面板下将 Class 设置为 aa。

【代码解析】

本实例关键功能是线段的绘制。下面就是这个知识点的详细讲解。

如果想要实现线段的绘制，首先要对线段的起始位置进行设置，需要使用 CGContext 的 CGContextMoveToPoint 函数。其语法形式如下：

```
void CGContextMoveToPoint (
    CGContextRef c,
    CGFloat x,
    CGFloat y
);
```

其中，CGContextRef c 表示上下文；CGFloat x 表示坐标中 x 的值；CGFloat y 表示坐标中 y 的值。然后再绘制线段的终点。使用的函数是 CGContextAddLineToPoint，其语法形式如下：

```
void CGContextAddLineToPoint (
    CGContextRef c,
    CGFloat x,
    CGFloat y
);
```

其中，CGContextRef c 表示上下文；CGFloat x 表示坐标中 x 的值；CGFloat y 表示坐标中 y 的值。在此代码中就是使用了 CGContextMoveToPoint 和 CGContextAddLineToPoint 函数使用了对量尺中线段的绘制。代码如下：

```
CGContextMoveToPoint(context, i, 0);
CGContextAddLineToPoint(context, i, 15);
```

实例 19 一笔画解答

【实例描述】

线段，可以绘制出各种各样的图形。本实例主要通过 Quartz2D 二维图形绘制引擎来实现绘制线段，从而来实现一笔画的效果。运行效果如图 1.54 所示。

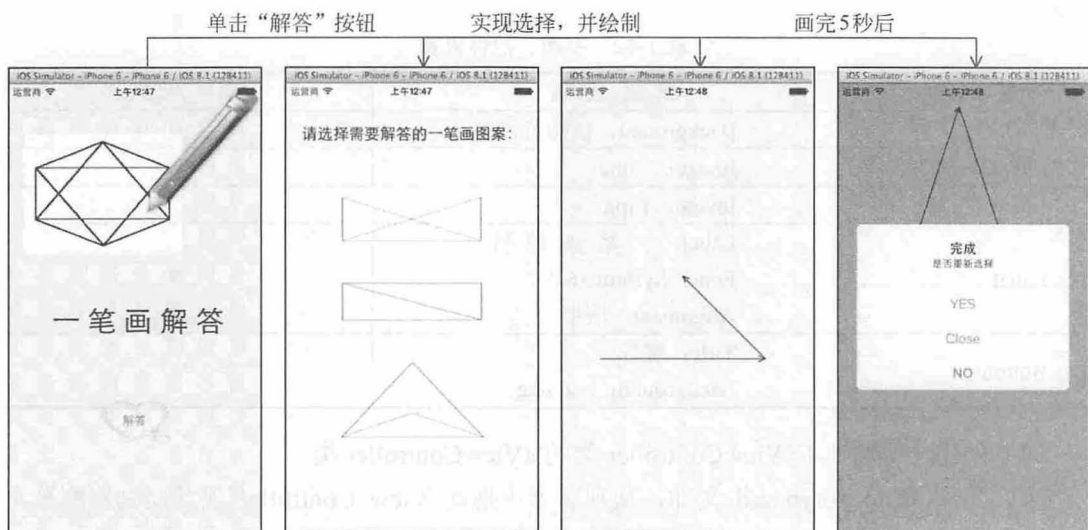


图 1.54 运行效果

【实现过程】

(1) 创建一个项目，命名为“一笔画解答”。

(2) 添加图片 1.jpg、2.png、3.png、4.png、5.png、6.png 到创建项目的 Supporting Files 文件夹中。

(3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 1.55 所示。

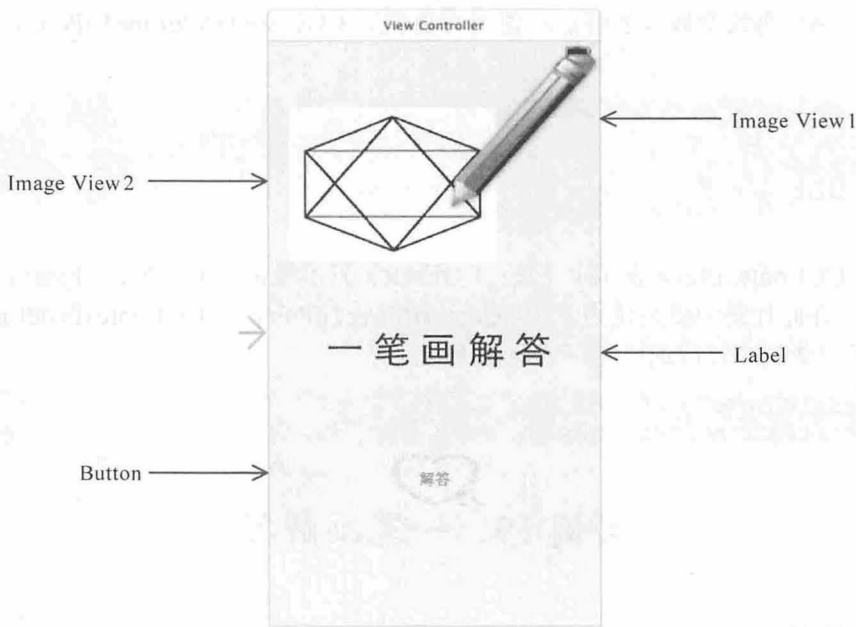


图 1.55 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-42 所示。

表 1-42 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 浅橘黄色	
Image View1	Image: 3.png	
Image View2	Image: 1.jpg	
Label	Label: 一笔画解答 Font: System 36.0 Alignment: 居中	
Button	Title: 解答 Background: ann.png	

(4) 创建一个基于 UIViewController 类的 aViewController 类。

(5) 回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aViewController。这时新增的 View Controller 视图控制器就变为了 A View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 aa，选中 Use Storyboard ID 复选框。

(6) 对 A View Controller 视图控制器的设计界面进行设计，效果如图 1.56 所示。

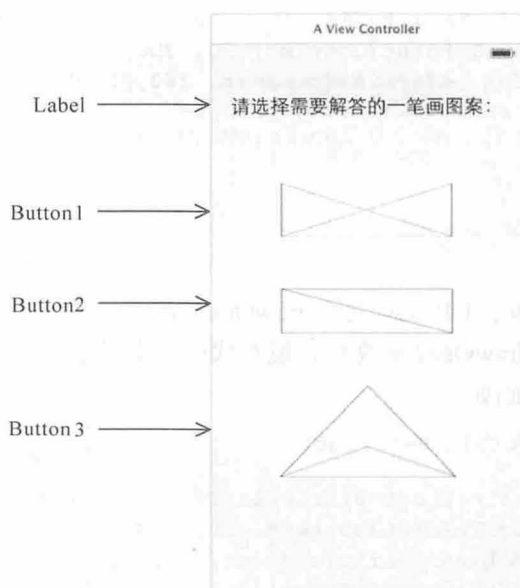


图 1.56 A View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设置如表 1-43 所示。

表 1-43 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅黄色	
Label	Text: 请选择需要解答的一笔画图案: Font: System 20.0 Alignment: 居中	
Button1	Title: (空) Background: 4.png	
Button2	Title: (空) Background: 5.png	
Button3	Title: (空) Background: 6.png	

(7) 创建一个基于 UIViewController 类的 aaViewController 类。

(8) 回到 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。

(9) 选择 Show the Identity inspector 图标，在 Custom Class 面板中，将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 View 面板下，将 Aa View Controller 的 Background 属性设置为浅绿色。它将实现 Button1 按钮所指向图形的一笔画简答效果。

(10) 创建一个基于 UIView 类的 drawOne1 类。

(11) 单击打开 drawOne1.m 文件，编写代码，实现绘制带箭头的线段，此线段是一条横向的直线段。程序代码如下：

```
- (void)drawRect:(CGRect)rect
{
```

```

CGContextRef context=UIGraphicsGetCurrentContext();
CGContextMoveToPoint(context, 50, 350);           //设置起点
CGContextAddLineToPoint(context, 260, 350);       //设置终点
CGContextStrokePath(context);
UILabel *a=[[UILabel alloc] initWithFrame:CGRectMake(250, 333, 30, 30)];
                                                    //实例化标签对象
a.text=@">";                                     //设置标签的文本内容
[self addSubview:a];
}

```

(12) 创建一个基于 UIView 类的 drawOne2 类。

(13) 单击打开 drawOne2.m 文件，编写代码，实现绘制带箭头的线段，此线段是一条垂直的线段。程序代码如下：

```

- (void)drawRect:(CGRect)rect
{
    CGContextRef context=UIGraphicsGetCurrentContext();
    CGContextMoveToPoint(context, 50, 350);           //设置起点
    CGContextAddLineToPoint(context, 50, 100);       //设置终点
    CGContextStrokePath(context);
    UILabel *a=[[UILabel alloc] initWithFrame:CGRectMake(45, 90, 30, 30)];
    a.text=@"^";                                     //设置标签的文本内容
    [self addSubview:a];
}

```

(14) 创建一个基于 UIView 类的 drawOne3 类。

(15) 单击打开 drawOne3.m 文件，编写代码，实现绘制带箭头的线段，此线段是一条垂直的线段。

(16) 创建一个基于 UIView 类的 drawOne4 类。

(17) 单击打开 drawOne4.m 文件，编写代码，实现绘制带箭头的线段，此线段是一条具有倾斜角的线段。程序代码如下：

```

- (void)drawRect:(CGRect)rect
{
    CGContextRef context=UIGraphicsGetCurrentContext();
    CGContextMoveToPoint(context, 260, 350);           //设置起点
    CGContextAddLineToPoint(context, 50, 100);       //设置终点
    CGContextStrokePath(context);
    UILabel *a=[[UILabel alloc] initWithFrame:CGRectMake(235, 327, 30, 30)];
    a.text=@"<";                                     //设置标签的文本内容
    a.transform=CGAffineTransformMakeRotation(225*3.14/180); //旋转
    [self addSubview:a];
}
@end

```

(18) 创建一个基于 UIView 类的 drawOne5 类。

(19) 单击打开 drawOne5.m 文件，编写代码，实现绘制带箭头的线段，此线段是一条具有倾斜角的线段。程序代码如下：

```

- (void)drawRect:(CGRect)rect
{
    CGContextRef context=UIGraphicsGetCurrentContext();
    CGContextMoveToPoint(context, 50, 350);           //设置起点
    CGContextAddLineToPoint(context, 260, 100);       //设置终点
    CGContextStrokePath(context);
}

```

```

UILabel *a=[[UILabel alloc]initWithFrame:CGRectMake(44,325 , 30, 30)];
a.text=@"<";
a.transform=CGAffineTransformMakeRotation(325*3.14/180);    //旋转
[self addSubview:a];
}

```

(20) 创建一个基于 UIViewController 类的 bbViewController 类。

(21) 打开 aaViewController.h 文件，编写代码，实现头文件以及对象的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
//头文件
#import "drawOne1.h"
#import "drawOne2.h"
#import "drawOne3.h"
#import "drawOne4.h"
#import "drawOne5.h"
#import "bbViewController.h"
@interface aaViewController : UIViewController{
    //对象
    drawOne1 *a;
    drawOne2 *b;
    drawOne3 *c;
    drawOne4 *d;
    drawOne5 *e;
    bbViewController *bvc;
}
@end

```

(22) 打开 aaViewController.m 文件，编写代码，实现初始化界面、Button1 图形的一笔画简答的动画效果以及警告视图的选择。使用的方法如表 1-44 所示。

表 1-44 aaViewController.m 文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
aa	一笔画的动画效果
bb	
cc	
dd	
ee	
gg	弹出警告视图
alertView:clickedButtonAtIndex:	响应警告视图的选择

其中，viewLoad 方法实现的是对界面进行的初始化实现。程序代码如下：

```

- (void)viewDidLoad
{
    //移除指定的视图对象
    [a removeFromSuperview];
    [b removeFromSuperview];
    [c removeFromSuperview];
    [d removeFromSuperview];
    [e removeFromSuperview];
    [self performSelector:@selector(aa) withObject:self afterDelay:3];
    //经过 3 秒后执行 aa 方法
}

```

```
[super viewDidLoad];
// Do any additional setup after loading the view.
}
```

aa、bb、cc、dd、ee 方法实现的是一笔画简答的动画效果，程序代码如下：

```
-(void)aa{
    a=[[drawOne1 alloc]initWithFrame:CGRectMake(0, 0, 320, 500)];
    [a setBackgroundColor:[UIColor clearColor]];           //设置背景颜色
    [self.view addSubview:a];
    [self performSelector:@selector(bb) withObject:self afterDelay:1];
                                                    //经过 1 秒后执行 bb 方法
}
-(void)bb{
    c=[[drawOne3 alloc]initWithFrame:CGRectMake(0, 0, 320, 500)];
    [c setBackgroundColor:[UIColor clearColor]];           //设置背景颜色
    [self.view addSubview:c];
    [self performSelector:@selector(cc) withObject:self afterDelay:1];
                                                    //经过 1 秒后执行 cc 方法
}
-(void)cc{
    e=[[drawOne5 alloc]initWithFrame:CGRectMake(0, 0, 320, 500)];
    [e setBackgroundColor:[UIColor clearColor]];           //设置背景颜色
    [self.view addSubview:e];
    [self performSelector:@selector(dd) withObject:self afterDelay:1];
                                                    //经过 1 秒后执行 dd 方法
}
-(void)dd{
    b=[[drawOne2 alloc]initWithFrame:CGRectMake(0, 0, 320, 500)];
    [b setBackgroundColor:[UIColor clearColor]];           //设置背景颜色
    [self.view addSubview:b];
    [self performSelector:@selector(ee) withObject:self afterDelay:1];
                                                    //经过 1 秒后执行 ee 方法
}
-(void)ee{
    d=[[drawOne4 alloc]initWithFrame:CGRectMake(0, 0, 320, 500)];
    [d setBackgroundColor:[UIColor clearColor]];           //设置背景颜色
    [self.view addSubview:d];
    [self performSelector:@selector(gg) withObject:self afterDelay:5];
                                                    //经过 5 秒后执行 gg 方法
}
```

gg 和 alertView:clickedButtonAtIndex:方法实现的是对弹出的警告视图的操作。程序代码如下：

```
//弹出警告视图
-(void)gg{
    UIAlertView *alert=[[UIAlertView alloc]initWithTitle:@"完成" message:
    @"是否查看下一个" delegate:self cancelButtonTitle:@"NO" otherButtonTitles:
    @"YES", nil];
    [alert show];
}
//实现响应
-(void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTextAtIndex:buttonIndex];
    //判断 str 是否为"NO"
    if([str isEqualToString:@"NO"]){
```



```
[self viewDidLoad];
}else{
    bvc=[self.storyboard instantiateViewControllerWithIdentifier:@"bb"];
    [self.view addSubview:bvc.view];           //添加视图控制器
}
}
```

以上就是第一种图形的一笔画简答的步骤，由于篇幅的限制，对于后两种的实现，请读者参考源代码。最后将界面出现的按钮和各个视图控制器中的设计界面进行关联，如表 1-45 所示。

表 1-45 按钮和各个视图控制器中的设计界面的关联

按 钮	关联的设计界面
Button	与A View Controller视图控制器关联
Button1	与Aa View Controller视图控制器关联
Button2	与Bb View Controller视图控制器关联
Button3	与Cc View Controller视图控制器关联

这时的画布效果如图 1.57 所示。

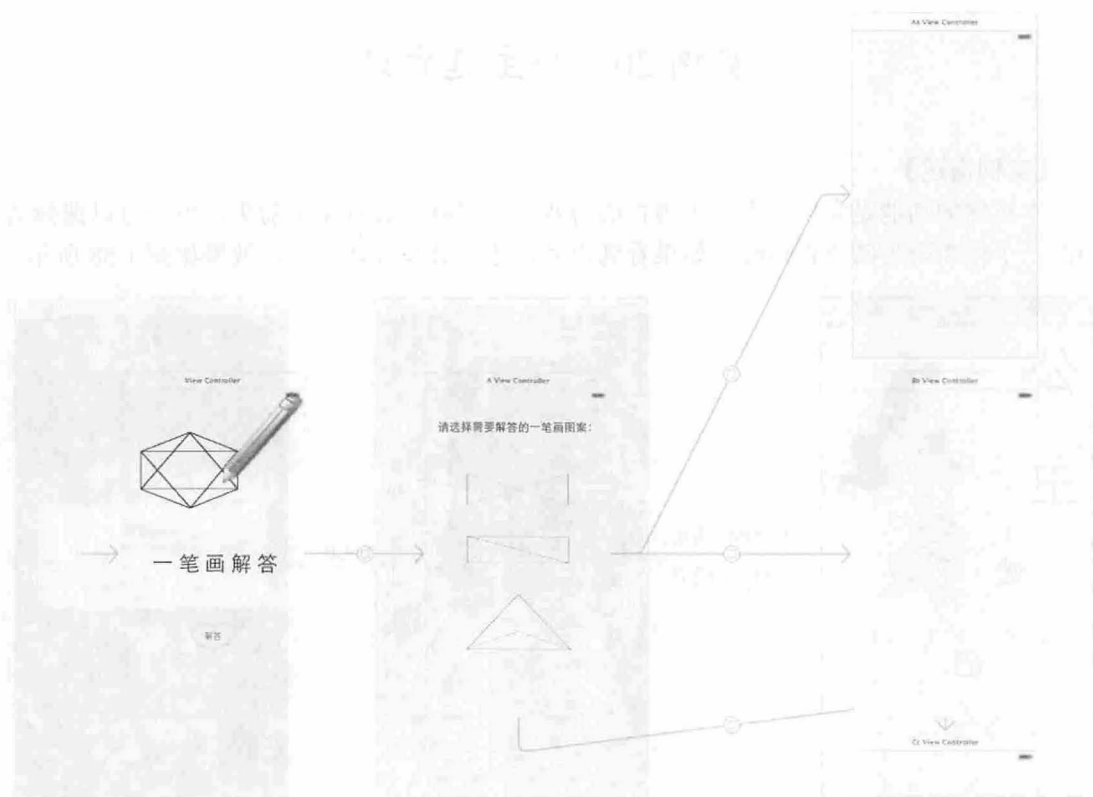


图 1.57 画布效果

【代码解析】

本实例关键功能是一笔画解答的实现。下面就是这个知识点的详细讲解。

在本实例中，要实现一笔画的解答，首先，需要将使用的线段先绘制好，然后，在一个视图控制器中，使用 `performSelector:withObject:afterDelay:` 方法将这些绘制的线段组合成动画，实现一笔画解答的动画效果，代码如下：

```

- (void)viewDidLoad
{
    .....
    [self performSelector:@selector(aa) withObject:self afterDelay:3];
    //经过 3 秒后执行 aa 方法
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}
- (void)aa{
    a=[[drawOne1 alloc] initWithFrame:CGRectMake(0, 0, 320, 500)];
    [a setBackgroundColor:[UIColor clearColor]]; //设置背景颜色
    [self.view addSubview:a];
    [self performSelector:@selector(bb) withObject:self afterDelay:1];
}

```

实例 20 公主逃亡记

【实例描述】

本实例的功能是实现一个公主逃亡的游戏。在屏幕上会有 4 个箭头，用户可以选择其中的一个作为公主的逃亡路线，如果看到户外，才算游戏成功。运行效果如图 1.58 所示。



图 1.58 运行效果

【实现过程】

- (1) 创建一个项目，命名为“公主逃亡记”。
- (2) 添加图片 1.png、2.png、3.png、4.png、5.jpg、6.jpg、7.png、8.jpg 到创建项目的

Supporting Files 文件夹中。

(3) 单击打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果 1.59 如图所示。

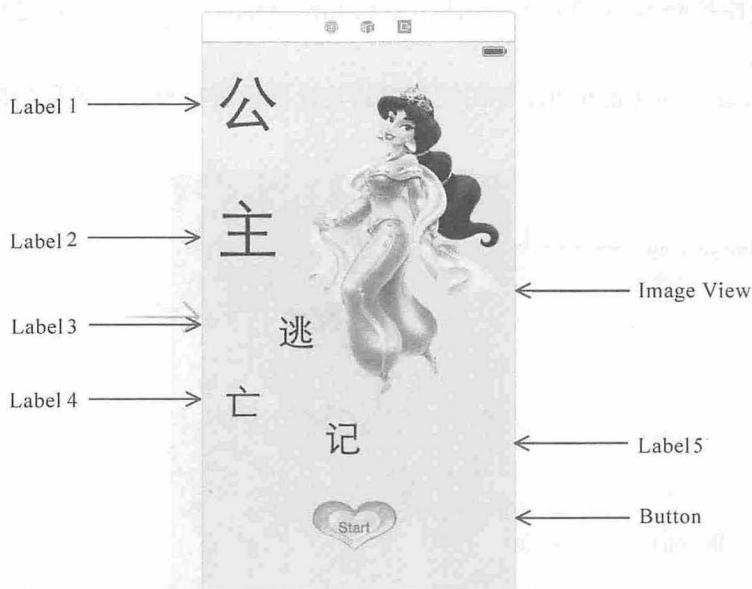


图 1.59 View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设计如表 1-46 所示。

表 1-46 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 浅紫色	
Image View	Image: 6.png	
Label1	Text: 公 Font: System 66.0 Alignment: 居中	
Label2	Text: 主 Font: System 66.0 Alignment: 居中	
Label3	Text: 逃 Font: System 38.0 Alignment: 居中	
Label4	Text: 亡 Font: System 38.0 Alignment: 居中	
Label5	Text: 记 Font: System 38.0 Alignment: 居中	
Button	Title: Start Background: 7.png	

(4) 创建一个基于 UIViewController 类的 aaViewController 类。

(5) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 aaViewController。这时新增的 View Controller 视图控制器就变为了 Aa View Controller 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 aa，选中 Use Storyboard ID 复选框。

(6) 对 Aa View Controller 视图控制器的设计界面进行设计，效果如图 1.60 所示。

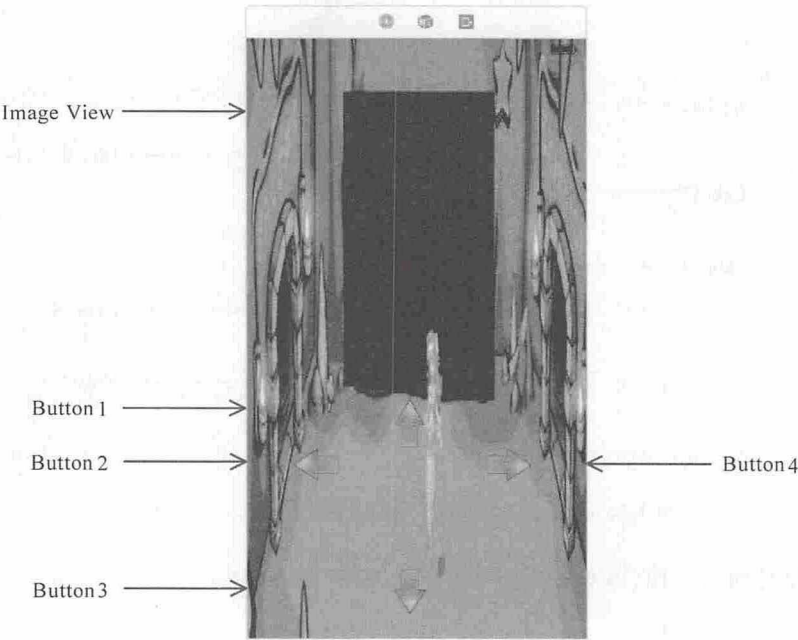


图 1.60 Aa View Controller 视图控制器的设计界面效果

需要添加的视图、控件以及对它们的设计如表 1-47 所示。

表 1-47 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: bg.jpg	
Button1	Title: (空) Image: 2.png	将此按钮和aaViewController.h文件进行 动作Up:的声明和关联
Button2	Title: (空) Image: 3.png	将此按钮和aaViewController.h文件进行 动作Left:的声明和关联
Button3	Title: (空) Image: 1.png	将此按钮和aaViewController.h文件进行 动作Down:的声明和关联
Button4	Title: (空) Image: 4.png	将此按钮和aaViewController.h文件进行 动作Right:的声明和关联

(7) 将 Start 按钮和 Aa View Controller 视图控制器的设计界面进行关联。

(8) 创建一个基于 UIViewController 类的 bbViewController、ccViewController、ddViewController 和 eeViewController 类。

(9) 回到 Main.storyboard 文件, 从视图库中拖动 4 个 View Controller 视图控制器到画布中。将 Class 设置为 bbViewController、ccViewController、ddViewController 和 eeViewController 类。将 Storyboard ID 分别设置为 bb、cc、dd、ee, 选中 Use Storyboard ID 复选框。

(10) 这些视图控制器的设计界面效果和 Aa View Controller 视图控制器的设计界面效果一样。需要添加的视图、控件以及对它们的设置也和 Aa View Controller 视图控制器的

(11) 打开 aaViewController.h 文件, 编写代码, 实现头文件, 以及对象的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import "bbViewController.h"           //头文件
#import "ccViewController.h"
@interface aaViewController : UIViewController{
    bbViewController *bvc;             //对象
    ccViewController *cvc;
}
//动作
- (IBAction)Up:(id)sender;            //从过渡层底部开始实现覆盖进入动画
- (IBAction)Down:(id)sender;         //从过渡层顶部开始实现覆盖进入动画
- (IBAction)Left:(id)sender;         //从过渡层左侧开始实现覆盖进入动画
- (IBAction)Right:(id)sender;        //从过渡层右侧开始实现覆盖进入动画
@end
```

(12) 打开 aaViewController.m 文件, 编写代码, 使用动画实现视图的切换。程序代码如下:

```
//从过渡层底部开始实现覆盖进入动画
- (IBAction)Up:(id)sender {
    CATransition *t=[CATransition animation];
    t.duration=2.0;
    t.type=kCATransitionMoveIn;        //设置动画的持续时间
    t.subtype=kCATransitionFromBottom;
    bvc=[self.storyboard instantiateViewControllerWithIdentifier:@"bb"];
    [self.view addSubview:bvc.view];   //添加视图对象
    [self.view.layer addAnimation:t forKey:@""];
}
//从过渡层顶部开始实现覆盖进入动画
- (IBAction)Down:(id)sender {
    CATransition *t=[CATransition animation];
    t.duration=2.0;
    t.type=kCATransitionMoveIn;        //设置动画类型
    t.subtype=kCATransitionFromTop;
    [self.view.layer addAnimation:t forKey:@""];
}
//从过渡层左侧开始实现覆盖进入动画
- (IBAction)Left:(id)sender {
    CATransition *t=[CATransition animation];
    t.duration=2.0;
    t.type=kCATransitionMoveIn;        //设置动画类型
```



```

t.subtype=kCATransitionFromLeft;
bvc=[self.storyboard instantiateViewControllerWithIdentifier:@"bb"];
[self.view addSubview:bvc.view];           //添加视图对象
[self.view.layer addAnimation:t forKey:@""];
}
//从过渡层右侧开始实现覆盖进入动画
- (IBAction)Right:(id)sender {
    CATransition *t=[CATransition animation];
    t.duration=2.0;
    t.type=kCATransitionMoveIn;
    t.subtype=kCATransitionFromRight;      //设置方向
    cvc=[self.storyboard instantiateViewControllerWithIdentifier:@"cc"];
    [self.view addSubview:cvc.view];
    [self.view.layer addAnimation:t forKey:@""];
}

```

在每一个视图控制器中都有 Up:、Down:、Left:、Right:方法，它们实现的功能都是一样的，就是实现视图在不同视图之间切换的功能，由于篇幅有限，这里给出读者一个方向执行图，如图 1.61 所示。对于这些方法的实现，请读者参考源代码。

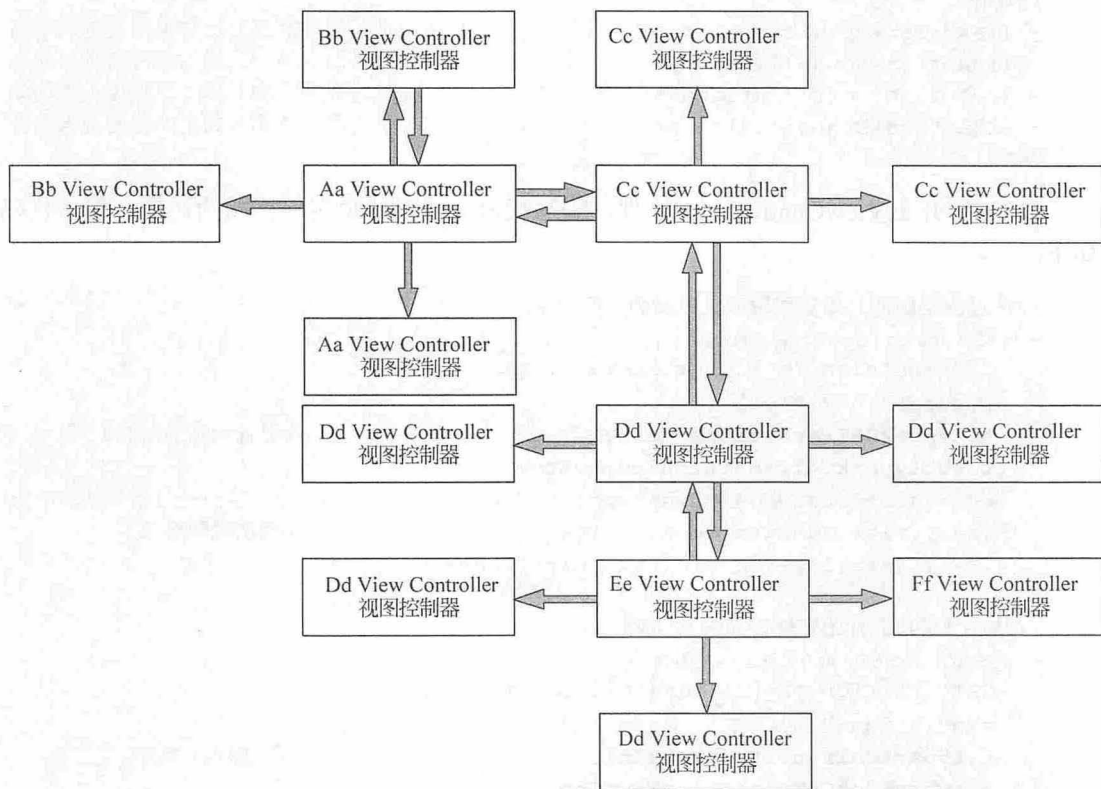


图 1.61 方向执行图

注意：在此图中，箭头表示方向。其中，在多处可以看到有返回上一层的箭头，以在 Bb View Controller 视图控件的界面中，单击上下的箭头就可以返回 Aa View Controller 的界面，它的实现代码如下：

```
- (IBAction)Down:(id)sender {
    CATransition *t=[CATransition animation];
    t.duration=2.0;
    t.type=kCATransitionMoveIn;
    t.subtype=kCATransitionFromTop;                //设置方向
    [self.view.superview.layer addAnimation:t forKey:@""];
    [self.view removeFromSuperview];                //移除指定的视图对象
}
```

【代码解析】

本实例关键功能是方向的改变。下面就是这个知识点的详细讲解。

方向的改变需要使用 CATransition 的 subtype 属性，在此代码中就是使用了 subtype 属性，在用户单击某一箭头后，就会在此方向上发生视图的切换。代码如下：

```
t.subtype=kCATransitionFromBottom;
```

第2章 图形图像（二）

图形图像与我们的生活是息息相关的。使用图形图像不仅可以制作万花筒，还可以实现照片墙以及游戏连连看的效果。本章将继续讲解与图形图像相关的实例。

实例 21 简易相框

【实例描述】

本实例主要实现的功能是绘制一个具有简易相框的图像视图。它将单调的图像视图变得更为丰富。运行效果如图 2.1 所示。

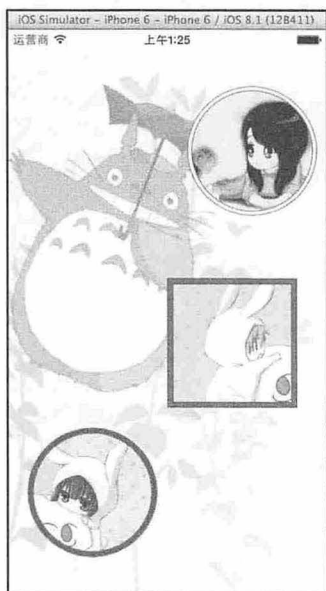


图 2.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“简易相框”。
- (2) 添加 1.jpg、2.jpg、3.jpg、4.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIImageView 类的 PathImageView 类。
- (4) 打开 PathImageView.h 文件，编写代码实现宏定义、数据结构的定义、对象、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
```

```

#define LINE_BORDER_WIDTH 1.0 //宏定义
//数据结构的定义
typedef enum {
    GBPathImageViewTypeCircle,
    GBPathImageViewTypeSquare
} PathImageViewType;
@interface PathImageView : UIImageView{
    UIImage *originalImage; //图像对象
}
//属性
@property (nonatomic) PathImageViewType pathType;
@property (nonatomic, strong) UIColor *borderColor;
@property (nonatomic, strong) UIColor *pathColor;
@property float pathWidth;
//方法
- (id)initWithFrame:(CGRect)frame image:(UIImage *)image;
//使用框架、图像进行初始化
- (id)initWithFrame:(CGRect)frame image:(UIImage *)image pathType:(PathImage
ViewType)pathType pathColor:(UIColor *)pathColor borderColor:(UIColor
*)borderColor pathWidth:(float)pathWidth;
- (void)draw; //绘制
@end

```

(5) 打开 PathImageView.m 文件，编写代码。实现对具有相框图像视图的绘制。使用的方法如表 2-1 所示。

表 2-1 PathImageView.m文件中方法总结

方 法	功 能
setDefaultParam	设置默认值
initWithFrame:image:	使用框架、图像进行初始化
initWithFrame:image:pathType:pathColor:borderColor: pathWidth:	使用框架、图像、路径类型、路径颜色、边框、边框颜色进行初始化
draw	绘制

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，setDefaultParam 方法实现对默认值的设置。代码如下：

```

-(void)setDefaultParam {
    _pathType = GBPathImageViewTypeCircle;
    _pathColor = [UIColor whiteColor];
    _borderColor = [UIColor darkGrayColor]; //边框颜色
    _pathWidth = 5.0;
}

```

initWithFrame:image:pathType:pathColor:borderColor:pathWidth:方法通过框架、图像、路径类型、路径颜色、边框、边框颜色等对具有相框的图像视图进行初始化。程序代码如下：

```

- (id)initWithFrame:(CGRect)frame image:(UIImage *)image pathType:(PathImage
ViewType)pathType pathColor:(UIColor *)pathColor borderColor:(UIColor *)
borderColor pathWidth:(float)pathWidth
{
    self = [super initWithFrame:frame];
    if (self) {
        originalImage = image;
    }
}

```

```

    _pathType = pathType; //设置路径类型
    _pathColor = pathColor;
    _borderColor = borderColor; //设置边框颜色
    _pathWidth = pathWidth;
    [self draw]; //绘制
}
return self;
}

```

draw 方法实现对具有相框的图像视图的绘制。程序代码如下：

```

-(void)draw {
    CGRect rect;
    rect.size = self.frame.size; //获取尺寸
    rect.origin = CGPointMake(0, 0);
    CGRect rectImage = rect;
    rectImage.origin.x += _pathWidth;
    rectImage.origin.y += _pathWidth;
    rectImage.size.width -= _pathWidth*2.0;
    rectImage.size.height -= _pathWidth*2.0;
    UIGraphicsBeginImageContextWithOptions(rect.size, 0, 0); //创建一个基于位图的上下文
    CGContextRef ctx = UIGraphicsGetCurrentContext(); //创建图形上下文
    //设置图像视图的形状
    switch (_pathType) {
        case GBPathImageViewTypeCircle:
            CGContextAddEllipseInRect(ctx, rect); //添加圆
            break;
        case GBPathImageViewTypeSquare:
            CGContextAddRect(ctx, rect); //添加矩形
            break;
        default:
            break;
    }
    CGContextClip(ctx);
    [originalImage drawInRect:rectImage]; //绘制
    //为路径的内侧和外侧添加边框
    rectImage.origin.x -= LINE_BORDER_WIDTH/2.0;
    rectImage.origin.y -= LINE_BORDER_WIDTH/2.0;
    rectImage.size.width += LINE_BORDER_WIDTH;
    rectImage.size.height += LINE_BORDER_WIDTH;
    rect.origin.x += LINE_BORDER_WIDTH/2.0;
    rect.origin.y += LINE_BORDER_WIDTH/2.0;
    rect.size.width -= LINE_BORDER_WIDTH;
    rect.size.height -= LINE_BORDER_WIDTH;
    CGContextSetStrokeColorWithColor(ctx, [_borderColor CGColor]); //设置线条的颜色
    CGContextSetLineWidth(ctx, LINE_BORDER_WIDTH); //设置线宽
    //绘制相框
    switch (_pathType) {
        case GBPathImageViewTypeCircle:
            CGContextStrokeEllipseInRect(ctx, rectImage); //绘制圆
            CGContextStrokeEllipseInRect(ctx, rect); //绘制圆
            break;
        case GBPathImageViewTypeSquare:
            CGContextStrokeRect(ctx, rectImage); //绘制矩形
            CGContextStrokeRect(ctx, rect); //绘制矩形
            break;
    }
}

```



```

        default:
            break;
    }
    float centerLineWidth = _pathWidth - LINE_BORDER_WIDTH*2.0;
    rectImage.origin.x -= LINE_BORDER_WIDTH/2.0+centerLineWidth/2.0;
    rectImage.origin.y -= LINE_BORDER_WIDTH/2.0+centerLineWidth/2.0;
    rectImage.size.width += LINE_BORDER_WIDTH+centerLineWidth;
    rectImage.size.height += LINE_BORDER_WIDTH+centerLineWidth;
    CGContextSetStrokeColorWithColor(ctx, [_pathColor CGColor]);
    //设置线条的颜色
    CGContextSetLineWidth(ctx, centerLineWidth); //绘制线宽
    //填充相框
    switch (_pathType) {
        case GBPathImageViewTypeCircle:
            CGContextStrokeEllipseInRect(ctx, rectImage); //绘制圆
            break;
        case GBPathImageViewTypeSquare:
            CGContextStrokeRect(ctx, rectImage); //绘制矩形
            break;
        default:
            break;
    }
    self.image = UIGraphicsGetImageFromCurrentImageContext(); //设置图像
    UIGraphicsEndImageContext(); //结束绘图,并关闭绘图环境
}

```

(6) 打开 ViewController, 编写代码, 实现 PathImageView.h 头文件的声明。

(7) 打开 Main.storyboard 文件, 对 View Controller 视图控件器的设计界面进行设计, 向其中添加 Image View 图像视图, 并将 Image 设置为 4.jpg。

(8) 打开 ViewController.m 文件, 编写代码, 实现具有相框的图像视图对象的创建与设置。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //创建默认的具有相框的图像视图对象
    PathImageView *squareImage1 = [[PathImageView alloc] initWithFrame:CGRectMake(180, 57, 130, 130) image:[UIImage imageNamed:@"1.jpg"]];
    [self.view addSubview:squareImage1];
    //创建具有正方形相框的图像视图对象
    PathImageView *squareImage2 = [[PathImageView alloc] initWithFrame:CGRectMake(160, 250, 130, 130) image:[UIImage imageNamed:@"3.jpg"] pathType:GBPathImageViewTypeSquare pathColor:[UIColor redColor] borderColor:[UIColor blueColor] pathWidth:6.0];
    [self.view addSubview:squareImage2];
    //创建具有圆形相框的图像视图对象
    PathImageView *squareImage3 = [[PathImageView alloc] initWithFrame:CGRectMake(20, 400, 130, 130) image:[UIImage imageNamed:@"2.jpg"] pathType:GBPathImageViewTypeCircle pathColor:[UIColor purpleColor] borderColor:[UIColor purpleColor] pathWidth:6.0];
    [self.view addSubview:squareImage3];
}

```

【代码解析】

本实例关键功能是相框的实现。下面就是这个知识点的详细讲解。

在本实例中，相框的实现使用了 `CGContext` 的 `CGContextAddEllipseInRect`、`CGContextStrokeEllipseInRect` 函数实现了图像视图和相框形状为圆形的绘制；`CGContextAddRect`、`CGContextStrokeRect` 函数实现了图像视图和相框形状为正方形的绘制。代码如下：

```
//设置图像视图的形状
switch (_pathType) {
    case GBPathImageViewTypeCircle:
        CGContextAddEllipseInRect(ctx, rect);
        break;
    case GBPathImageViewTypeSquare:
        CGContextAddRect(ctx, rect);
        break;
    default:
        break;
}
CGContextClip(ctx);
[originalImage drawInRect:rectImage];
.....

//绘制相框
switch (_pathType) {
    case GBPathImageViewTypeCircle:
        CGContextStrokeEllipseInRect(ctx, rectImage);
        CGContextStrokeEllipseInRect(ctx, rect);
        break;
    case GBPathImageViewTypeSquare:
        CGContextStrokeRect(ctx, rectImage);
        CGContextStrokeRect(ctx, rect);
        break;
    default:
        break;
}
.....

//填充相框
switch (_pathType) {
    case GBPathImageViewTypeCircle:
        CGContextStrokeEllipseInRect(ctx, rectImage);
        break;
    case GBPathImageViewTypeSquare:
        CGContextStrokeRect(ctx, rectImage);
        break;
    default:
        break;
}
```

实例 22 图 像 滤 镜

【实例描述】

在 PS 中经常看到为图片添加怀旧效果或者黑白效果，这些效果都是通过滤镜实现的。滤镜不仅可以在 PS 中使用，它也可以使用在 iOS 的手机中。本实例就为读者实现如何在图像上添加滤镜的功能。当用户选择自定义选择器中的一个滤镜效果后，图像视图上就会添加此滤镜效果。运行效果如图 2.2 所示。



图 2.2 运行效果

【实现过程】

- (1) 创建一个项目，命名为“图像滤镜”。
- (2) 添加图片 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现插座变量、对象以及遵守协议的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController<UIPickerViewDataSource,
UIPickerViewDelegate>{
    IBOutlet UIImageView *imageView;           //图像视图的插座变量
    UIImage *orgImage;
    NSArray *items;
}
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 2.3 所示。

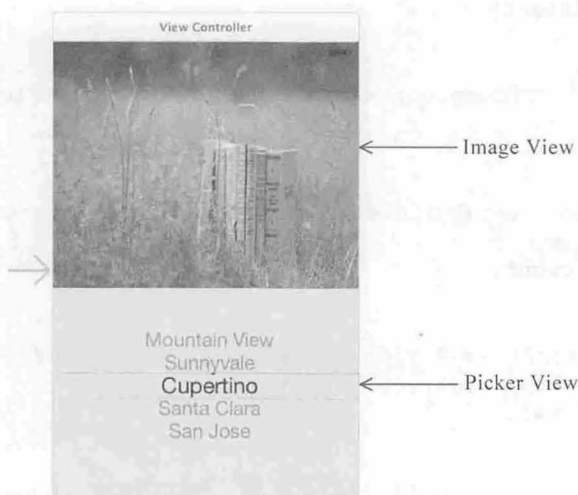


图 2.3 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-2 所示。

表 2-2 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	
Image View	Image View: 1.jpg	与插座变量imageView关联
Picker View		与dataSource关联 与delegate关联

(5) 打开 ViewController.m 文件, 编写代码, 实现选择器的内容填充以及选择行后的滤镜效果。使用的方法如表 2-3 所示。

表 2-3 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
numberOfComponentsInPickerView:	获取块数
pickerView:numberOfRowsInComponent:	获取行数
pickerView:titleForRow:forComponent:	获取内容
pickerView:didSelectRow:inComponent:	响应选择的行

其中, 选择器内容的填充需要使用 viewDidLoad、numberOfComponentsInPickerView:、pickerView:numberOfRowsInComponent:、pickerView:titleForRow:forComponent:方法, 代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    orgImage = [UIImage imageNamed:@"1.jpg"];
    imageView.image = orgImage; //设置图像视图显示的图像
    items = @[@"Original",@"CIColorToSRGBToneCurve",@"CIPhotoEffectChrome",
    @"CIPhotoEffectFade",@"CIPhotoEffectInstant",@"CIPhotoEffectMono",
    @"CIPhotoEffectNoir",@"CIPhotoEffectProcess",@"CIPhotoEffectTonal",
    @"CIPhotoEffectTransfer",@"CISRGBToneCurveToLinear",
    @"CIVignetteEffect"];
}
//获取块数
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 1;
}
//获取行数
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:
(NSInteger)component {
    return items.count;
}
//获取内容
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)
row forComponent:(NSInteger)component {
    return items[row];
}

```

响应选择器选中的行, 从而实现滤镜的改变, 需要使用 pickerView:didSelectRow:inComponent:方法。程序代码如下:

```

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger) row
inComponent:
(NSInteger)component {
    //判断 row 是否为 0
    if (row == 0) {
        imageView.image = orgImage;           //设置图像视图显示的图像
        return;
    }
    //实例化对象
    CIImage *ciImage = [[CIImage alloc] initWithImage:orgImage];
    CIContext *context = [CIContext contextWithOptions:nil];
    CIFilter *filter = [CIFilter filterWithName:items[row] keysAndValues:
    kCIInputImageKey, ciImage, nil];
    [filter setDefaults];                      //设置过滤器中所有的输入值为默认值
    //得到过滤后的图片
    CIImage *outputImage = [filter outputImage];
    //转换图片
    CGImageRef cgImage = [context createCGImage:outputImage fromRect:
    [outputImage extent]];
    imageView.image = [UIImage imageWithCGImage:cgImage];
    //设置图像视图显示的图像
    CGImageRelease(cgImage);
}

```

【代码解析】

本实例关键功能是图片的滤镜效果。下面就是这个知识点的详细讲解。要实现滤镜效果，需要 4 个步骤：

1. 创建一个CIImage对象

为了对图片数据进行存放，需要创建一个 CIImage 对象，在本实例中此对象的创建使用了 initWithImage:方法，代码如下：

```
CIImage *ciImage = [[CIImage alloc] initWithImage:orgImage];
```

2. 创建一个CIContext对象

要对 Core Image 图片进行处理都必须通过 CIContext 进行。此时，就需要创建一个 CIContext 对象，在本实例中此对象的创建需要使用 CIContext 的 contextWithOptions:方法，代码如下：

```
CIContext *context = [CIContext contextWithOptions:nil];
```

3. 创建一个CIFilter对象

要实现图片的滤镜，必须要创建一个 CIFilter 对象即过滤器，需要使用 CIFilter 的 filterWithName:keysAndValues:方法。其语言形式如下：

```
+ (CIFilter *)filterWithName:(NSString *)name keysAndValues:key0, ...;
```

其中，(NSString *)name 表示滤镜的名称；key0, ...表示滤镜属性的键和值。在本实例中，就是使用 filterWithName:keysAndValues:方法实现了对 CIFilter 对象的创建以及属性的设置，代码如下：

```
CIFilter *filter = [CIFilter filterWithName:items[row] keysAndValues:
```



```
kCIInputImageKey, ciImage, nil];
```

其中，items[row]表示滤镜名称；kCIInputImageKey, ciImage 表示滤镜属性的键和值。

4. 输出滤镜

最后一步就是要输出滤镜了，这时需要使用 outputImage 属性实现。代码如下：

```
CIImage *outputImage = [filter outputImage];
```

这里需要注意输出的滤镜是 CIImage 形式的，需要将它转换为 UIImage 形式的图像才可以显示，需要使用 UIImage 的 initWithCGImage:方法，其语法形式如下：

```
+ (UIImage *)initWithCGImage:(CGImageRef)cgImage;
```

其中，(CGImageRef)cgImage 表示 Quartz 图像对象。在此代码中使用了 initWithCGImage:方法将 CIImage 转换为 UIImage，代码如下：

```
CGImageRef cgImage = [context createCGImage:outputImage fromRect:
[outputImage extent]];
imageView.image = [UIImage initWithCGImage:cgImage];
```

实例 23 图像的点击放大

【实例描述】

本实例实现的是点击放大的功能，它有点像查看 QQ 的图像一样。当点击图像后，此方法就会放大图像。运行效果如图 2.4 所示。

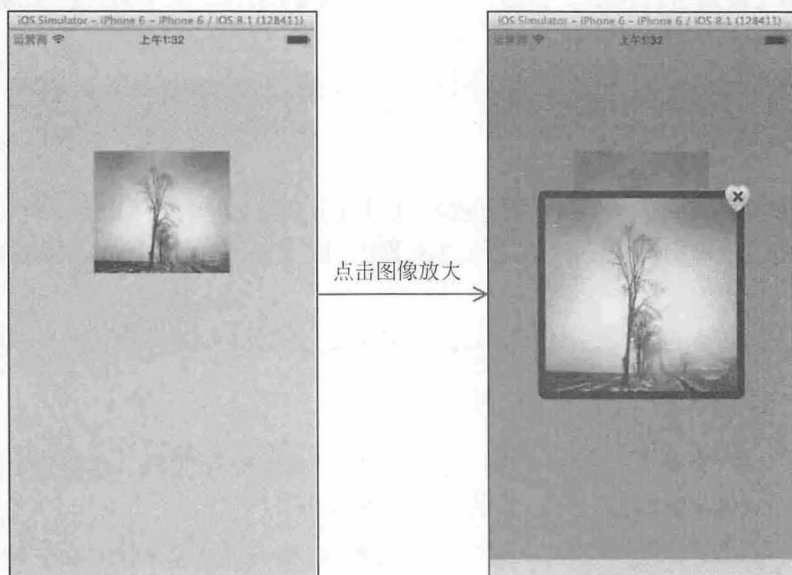


图 2.4 运行效果

【实现过程】

(1) 创建一个项目，命名为“点击放大”。

(2) 添加图像 1.jpg、2.png 到创建项目的 Supporting Files 文件夹中。

(3) 打开 ViewController.h 文件，编写代码，实现宏定义、插座变量、对象的声明。

程序代码如下：

```
#import <UIKit/UIKit.h>
#define BIG_IMG_WIDTH 200.0
#define BIG_IMG_HEIGHT 200.0
@interface ViewController : UIViewController{
    IBOutlet UIImageView *Picture;           //声明插座变量
    UIView *background;                     //声明对象
}
@end
```

(4) 打开 Main.storyboard 文件，将设计界面的背景颜色设置为灰色，拖动 Image View 图像视图到设计界面中，将 Image 设置为 1.jpg。将此视图和插座变量 Picture 关联。

(5) 打开 ViewController.m 文件，编写代码，实现图像点击放大的功能。使用的方法如表 2-4 所示。

表 2-4 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
fangda	放大
shakeToShow:	放大过程中出现的缓慢动画
suoxiao	缩小

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewDidLoad 方法在视图加载后调用，实现对图像视图的设置以及轻拍手势识别器对象的创建。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    Picture.userInteractionEnabled = YES;
    UITapGestureRecognizer *singleTap = [[UITapGestureRecognizer alloc]
    initWithTarget:self action:@selector(fangda)]; //创建轻拍手势识别器对象
    [Picture addGestureRecognizer:singleTap];
}
```

fangda 方法实现单击点击图像后图像的放大功能。程序代码如下：

```
- (void)fangda
{
    //创建灰色透明背景，使其背后内容不可操作
    UIView *bgView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 320,
    548)];
    background = bgView;
    [bgView setBackgroundColor:[UIColor colorWithRed:0.3 green:0.3 blue:0.3
    alpha:0.7]]; //设置背景颜色
    [self.view addSubview:bgView];
    //创建边框视图
```

```

UIView *borderView = [[UIView alloc] initWithFrame:CGRectMake(0,
0,BIG_IMG_WIDTH+16, BIG_IMG_HEIGHT+16)];
//将图层的边框设置为圆脚
borderView.layer.cornerRadius = 8; //设置圆角半径
borderView.layer.masksToBounds = YES;
//给图层添加一个有色边框
borderView.layer.borderWidth = 8; //设置边框的宽度
borderView.layer.borderColor = [[UIColor blackColor]CGColor];
[borderView setCenter:bgView.center];
//设置中心位置
[bgView addSubview:borderView];
//创建关闭按钮
UIButton *closeBtn = [UIButton buttonWithTypeCustom];
[closeBtn setImage:[UIImage imageNamed:@"2.png"] forState:UIControlStateNormal]; //设置按钮的图像
[closeBtn addTarget:self action:@selector(suoxiao) forControlEvents:
UIControlEventsTouchUpInside];
NSLog(@"borderview is %@",borderView);
[closeBtn setFrame:CGRectMake(borderView.frame.origin.x+borderView.frame.
size.width-20, borderView.frame.origin.y-6, 26, 27)]; //设置框架
[bgView addSubview:closeBtn];
//创建显示图像视图
UIImageView *imgView = [[UIImageView alloc] initWithFrame:CGRectMake(8,
8,BIG_IMG_WIDTH, BIG_IMG_HEIGHT)];
[imgView setImage:[UIImage imageNamed:@"1.jpg"]]; //设置图像视图的图像
[borderView addSubview:imgView];
[self shakeToShow:borderView]; //放大过程中的动画
//动画效果
CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
[UIView beginAnimations:nil context:context];
[UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
[UIView setAnimationDuration:2.6]; //设置动画的持续时间
[self.view exchangeSubviewAtIndex:0 withSubviewAtIndex:1];
[UIView setAnimationDelegate:bgView];
//设置委托
// 动画完毕后调用 animationFinished
[UIView commitAnimations];
}

```

shakeToShow:方法实现在放大过程中出现的缓慢动画。程序代码如下:

```

- (void)shakeToShow:(UIView*)aView{
    CAKeyframeAnimation* animation = [CAKeyframeAnimation animationWith
    KeyPath:@"transform"];
    animation.duration = 0.5; //设置动画持续时间
    NSMutableArray *values = [NSMutableArray array];
    //为数组添加对象
    [values addObject:[NSValue valueWithCATransform3D:CATransform3DMake
    Scale(0.1, 0.1, 1.0)]];
    [values addObject:[NSValue valueWithCATransform3D:CATransform3DMake
    Scale(1.0, 1.0, 1.0)]];
    animation.values = values;
    [aView.layer addAnimation:animation forKey:nil]; //添加动画
}

```

【代码解析】

本实例关键功能是点击图像后放大。下面就是这个知识点的详细讲解。

在本实例中点击图像后放大,首先需要创建一个轻拍的手势识别器,并使用 `addGestureRecognizer:` 方法添加到图像中,这样就可以实现点击的功能。代码如下:

```
UITapGestureRecognizer *singleTap = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(fangda)];
[Picture addGestureRecognizer:singleTap];
```

然后可以在 `fangda` 方法中实现图像放大效果,此时需要重新创建一个用于存放放大图像的图像视图。代码如下:

```
UIImageView *imgView = [[UIImageView alloc] initWithFrame:CGRectMake(8, 8,
BIG_IMG_WIDTH, BIG_IMG_HEIGHT)];
[imgView setImage:[UIImage imageNamed:@"1.jpg"]];
[borderView addSubview:imgView];
```

实例 24 万花筒

【实例描述】

相信万花筒对于很多读者都曾有一份美好的童年记忆。当玩家旋转万花筒的筒部时,就会看到各种变化的花型。那么,如何将万花筒变为一个在手机上的应用程序呢?本实例就会为读者去实现这么一个万花筒的功能,帮读者重拾童年的美好记忆。当用户调整控制面板中的滑块时,看到的形状是不一样的。运行效果如图 2.5 所示。



图 2.5 运行效果

【实现过程】

(1) 创建一个项目,命名为“万花筒”。

- (2) 添加图片 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UISlider 类的 Slider 类。
- (4) 打开 Slider.h 文件，编写代码，实现对象、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface Slider : UISlider{
    UILabel *valueLabel;                                //声明对象
}
@property (nonatomic) NSString *attributeKey;           //属性
- (void)setupWithAttributes:(NSDictionary*)attributes forKey:(NSString*)
attrKey;                                               //方法
@end
```

- (5) 打开 Slider.m 文件，编写代码，实现对滑块矩形的获取和对属性的设置。程序代码如下：

```
- (CGRect)thumbRectForBounds:(CGRect)bounds trackRect:(CGRect)rect value:
(float)value{
    CGRect result = [super thumbRectForBounds:bounds trackRect:rect value:
value];
    return result;
}
- (void)setupWithAttributes:(NSDictionary*)attributes forKey:(NSString*)
attrKey{
    NSDictionary *attr = attributes[attrKey];
    self.minimumValue = [attr[kCIAttributeSliderMin] floatValue]; //最小值
    self.maximumValue = [attr[kCIAttributeSliderMax] floatValue]; //最大值
    self.value = [attr[kCIAttributeDefault] floatValue];          //当前值
    self.attributeKey = attrKey;
}
```

- (6) 打开 ViewController.h 文件，编写代码，实现头文件、对象、实例变量、插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "Slider.h"
@interface ViewController : UIViewController{
    //对象
    CIFilter *filter;
    CGRect imageRect;
    //插座变量
    IBOutlet UIImageView *imageView;
    IBOutlet Slider *sizeSlider;
    IBOutlet Slider *rotationSlider;
    IBOutlet Slider *decaySlider;
    IBOutlet UIButton *button;
}
- (IBAction)show:(id) sender; //单击按钮后改变
@end
```

- (7) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 2.6 所示。



图 2.6 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-5 所示。

表 2-5 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 1.jpg	与插座变量imageView关联
Button		与插座变量button关联 与动作show关联
View		与插座变量vv关联
Slider1		与插座变量sizeSlider关联
Slider2		与插座变量rotationSlider关联
Slider3		与插座变量decaySlider关联
Label1	Text: 控件面板 Font: 32.0 Alignment: 居中	
Label2	Text: 尺寸	
Label3	Text: 旋转	
Label4	Text: 亮度	

(8) 打开 ViewController.m 文件，编写代码，实现当滑动滑块时，万花筒的变化。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    vv.layer.cornerRadius =10;
    button.layer.cornerRadius =10;           //按钮的圆角半径
    imageView.layer.cornerRadius =20;       //图像视图的圆角半径
    filter = [CIFilter filterWithName:@"CITriangleKaleidoscope"];
                                           //实例化对象
    [sizeSlider setupWithAttributes:filter.attributes forKey:@"inputSize"];
}

```

```

[rotationSlider setupWithAttributes:filter.attributes forKey:@"input
Rotation"];
[decaySlider setupWithAttributes:filter.attributes forKey:@"inputDecay"];
[filter setValue:[UIImage imageWithCGImage:imageView.image.CGImage]
forKey:kCIInputImageKey];
CGSize size = imageView.image.size; //获取尺寸
[filter setValue:[CIVector vectorWithX:size.width/2 Y:size.height/2]
forKey:@"inputPoint"];
imageRect = CGRectMake(0, 0, size.width, size.height);
}
//单击按钮后改变
- (IBAction)show:(id)sender {
    //实例化对象
    NSString *a1=[NSString stringWithFormat:@"%f",sizeSlider.value];
    NSString *a2=[NSString stringWithFormat:@"%f",rotationSlider.value];
    NSString *a3=[NSString stringWithFormat:@"%f",decaySlider.value];
    //设置过滤器的值
    [filter setValue:a1 forKey:sizeSlider.attributeKey];
    [filter setValue:a2 forKey:rotationSlider.attributeKey];
    [filter setValue:a3 forKey:decaySlider.attributeKey];
    UIImage *result = filter.outputImage;
    //判断 result 是否不为空
    if (result) {
        CIContext *context = [CIContext contextWithOptions:nil];
        CGImageRef cgImage = [context createCGImage:result fromRect: imageRect];
        imageView.image = [UIImage imageWithCGImage:cgImage];
        //设置图像视图显示的图像
        CGImageRelease(cgImage);
    } else {
        NSLog(@"warning: nil result");
    }
}
}

```

【代码解析】

本实例关键功能是图片的改变。下面就是这个知识点的详细讲解。

要想改变图像的大小、旋转等，需要将 `CIFilter` 对象的属性进行调整，在此代码中使用了 `setValue:forKey:` 方法，代码如下：

```

[filter setValue:a1 forKey:sizeSlider.attributeKey];
[filter setValue:a2 forKey:rotationSlider.attributeKey];
[filter setValue:a3 forKey:decaySlider.attributeKey];

```

最后需要将改变后的 Quartz 图像对象改变为 `UIImage` 图像进行显示。

实例 25 浏览商品图片

【实例描述】

在网上购物时，商家为了让用户能看清所选商品的细节，会提供照片的局部放大功能，当用户移动图片中的指示器时，就会出现一个相应的大图片。它显示的位置就是用户使用指示器指向的地方。本实例就是实现这么一个功能。在界面上会出现两个图像，其中一个是小的图像，另一个是较大的图像。当用户在小的图像上移动指示器时，会在较大的图像

中显示指示器所指示的位置。当然此程序不止这一个功能，当用户移动大图时，小图中的指示器会指向大图中显示的位置。运行效果如图 2.7 所示。



图 2.7 运行效果

【实现过程】

- (1) 创建一个项目，命名为“浏览商品图片”。
- (2) 添加图片 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 View 类。
- (4) 打开 View.h 文件，编写代码，实现宏定义、对象、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
//宏定义
#define HEIGHT [UIScreen mainScreen].bounds.size.height
#define WIDTH [UIScreen mainScreen].bounds.size.width
@interface View : UIView<UIScrollViewDelegate>{
    //声明对象
    UIView *zoomedView;
    UIView *miniMe;
    UIImageView *miniMeImageView;
    UIView *miniMeIndicator;
}
//属性
@property (nonatomic, strong) UIScrollView *scrollView;
@property (assign) NSInteger ratio;
-(View *)initWithView:(UIView *)viewToMap withRatio:(NSInteger)ratio;
    //初始化方法
@end
```

- (5) 打开 View.m 文件，编写代码，实现绘制商品浏览中小的视图以及实现浏览功能。使用的方法如表 2-6 所示。

表 2-6 View.m文件中方法总结

方 法	功 能
<code>initWithView:withRatio:</code>	初始化
<code>dragBegan:</code>	拖曳
<code>captureScreen:</code>	截图
<code>scrollViewDidScroll:</code>	滚动
<code>viewForZoomingInScrollView:</code>	获取放大后的视图
<code>scrollViewDidZoom:</code>	设置指示器的框架

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。绘制商品浏览中小的视图需要使用 `initWithView:withRatio:`、`captureScreen:`方法。其中，`initWithView:withRatio:`方法用来实现初始化的功能。

```

-(View *)initWithView:(UIView *)viewToMap withRatio:(NSInteger)ratio
{
    self = [super initWithFrame:viewToMap.frame];
    if (self)
    {
        zoomedView = viewToMap;
        _ratio = ratio;
        [self setBackgroundColor:[UIColor redColor]];
        //创建并设置滚动视图对象
        self.scrollView = [[UIScrollView alloc] initWithFrame:CGRectMake(0,
            0, 320, HEIGHT)];
        self.scrollView.contentSize = CGSizeMake(viewToMap.bounds.size.
            width, viewToMap.bounds.size.height); //设置滚动视图中内容视图的尺寸
        self.scrollView.delegate = self;
        self.scrollView.minimumZoomScale = 1;
        self.scrollView.maximumZoomScale = 20; //设置滚动视图可以缩放的最大值
        [self.scrollView setBounces:NO];
        [self.scrollView addSubview:viewToMap]; //添加视图对象到滚动视图中
        [self addSubview:self.scrollView];
        //创建并设置一个小的视图
        miniMe = [[UIView alloc] initWithFrame:CGRectMake(10, 10, viewToMap.
            frame.size.width/_ratio, viewToMap.frame.size.height/_ratio)];
        //实例化对象
        miniMe.clipsToBounds = YES; //剪裁超出父视图范围的子视图部分
        miniMeImageView = [[UIImageView alloc] initWithImage:[self capture
            Screen:viewToMap]];
        miniMeImageView.frame = CGRectMake(0, 0, miniMe.frame.size.width,
            miniMe.frame.size.height);
        [miniMe addSubview:miniMeImageView];
        self.scrollView.zoomScale = 5; //设置当前的缩放值
        [miniMe setBackgroundColor:[UIColor blueColor]];
        [self addSubview:miniMe];
        //创建并设置空白视图对象，作为指示器
        miniMeIndicator = [[UIView alloc] initWithFrame:CGRectMake(
            self.scrollView.contentOffset.x/_ratio/self.scrollView.zoomScale,
            self.scrollView.contentOffset.y/_ratio/self.scrollView.zoomScale,
            miniMe.frame.size.width/self.scrollView.zoomScale, miniMe.frame.
            size.height/self.scrollView.zoomScale)];
        [miniMeIndicator setBackgroundColor:[UIColor redColor]];
        //设置背景颜色
        [miniMeIndicator setAlpha:0.40];
        [miniMe addSubview:miniMeIndicator];
    }
}

```

```

//创建并设置按钮对象
UIButton *miniMeSelectorBtn = [UIButton buttonWithType:UIButton
TypeCustom];
miniMeSelectorBtn.frame = CGRectMake(0, 0, miniMe.frame.size.width,
miniMe.frame.size.height);
miniMeSelectorBtn.backgroundColor=[UIColor blackColor];
//设置背景颜色

[miniMeSelectorBtn addTarget:self action:@selector(dragBegan:
withEvent:) forControlEvents: UIControlEventTouchDragInside |
UIControlEventTouchDown];
//添加动作
miniMeSelectorBtn.clipsToBounds = YES;
[miniMe addSubview:miniMeSelectorBtn];
}
return self;
}

```

captureScreen:方法实现对屏幕的截取, 程序代码如下:

```

- (UIImage*) captureScreen: (UIView*) viewToCapture
{
    CGRect rect = [viewToCapture bounds];
    UIGraphicsBeginImageContextWithOptions(rect.size, YES, 0.0f);
    //创建位图的上下文, 并设置缩放因子
    [viewToCapture.layer renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    //返回一个基于当前图形上下文的图片
    UIGraphicsEndImageContext();
    //结束绘图, 并关闭绘图环境
    return image;
}

```

实现图像的浏览功能, 需要使用 **dragBegan:**、**scrollViewDidScroll:**、**viewForZoomingInScrollView:**、**scrollViewDidZoom:**方法。其中, **dragBegan:**方法实现拖曳功能, 从而改变指示器和滚动视图的滚动位置。程序代码如下:

```

- (void) dragBegan: (UIControl *)c withEvent:ev
{
    UITouch *touch = [[ev allTouches] anyObject];
    CGPoint touchPoint = [touch locationInView:miniMe]; //获取触摸点
    //判断当前的触摸点的 x 是否小于 0
    if(touchPoint.x<0)
    {
        touchPoint.x=0;
    }
    //判断当前的触摸点的 y 是否小于 0
    if(touchPoint.y<0)
    {
        touchPoint.y=0;
    }
    if(touchPoint.y + miniMe.frame.size.height/self.scrollView.zoomScale >
miniMe.frame.size.height)
    {
        //重新设置 touchPoint 即当前触摸点的 y 值
        touchPoint.y = miniMe.frame.size.height - miniMe.frame.size.height/
self.scrollView.zoomScale;
    }
    if(touchPoint.x + miniMe.frame.size.width/self.scrollView.zoomScale >
miniMe.frame.size.width)
    {
        //重新设置 touchPoint 即当前触摸点的 x 值
    }
}

```



```

        touchPoint.x = miniMe.frame.size.width - miniMe.frame.size.width/
        self.scrollView.zoomScale;
    }
    //设置框架
    miniMeIndicator.frame = CGRectMake(touchPoint.x, touchPoint.y, miniMe.
    frame.size.width/self.scrollView.zoomScale,miniMe.frame.size.height
    /self.scrollView.zoomScale);
    [self.scrollView setContentOffset:CGPointMake(touchPoint.x*_ratio*self.
    scrollView.zoomScale, touchPoint.y*_ratio*self.scrollView.zoomScale) animated:
    NO]; //设置可滚动区域的偏移量
}

```

scrollViewDidScroll:方法对指示器框架的设置程序代码如下:

```

- (void)scrollViewDidScroll:(UIScrollView *)scrollView
{
    miniMeIndicator.frame=CGRectMake(self.scrollView.contentOffset.x/_
    ratio/self.scrollView.zoomScale,
    self.scrollView.contentOffset.y/_ratio/self.scrollView.zoomScale,miniMe
    Indicator.frame.size.width,miniMeIndicator.frame.size.height); //设置框架
}

```

(6) 打开 ViewController.h 文件, 编写代码, 实现头文件、宏定义、对象的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "View.h"
//宏定义
#define WIDTH [UIScreen mainScreen].bounds.size.width
#define HEIGHT [UIScreen mainScreen].bounds.size.height
#define BUTTON_SIZE 10
@interface ViewController : UIViewController{
    View *MeView; //声明对象
}
@end

```

(7) 打开 ViewController.m 文件, 编写代码, 实现商品浏览的功能。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    UIView *tempView = [[UIView alloc] initWithFrame:self.view.frame];
    UIImageView *imgView = [[UIImageView alloc] initWithImage:[UIImage
    imageNamed:@"9.jpg"]];
    imgView.frame = tempView.frame; //设置框架
    [imgView setContentMode:UIViewContentModeScaleAspectFill]; //设置图像的显示模式
    [tempView addSubview:imgView];
    MeView = [[View alloc] initWithView:tempView withRatio:4];
    [self.view addSubview:MeView];
}

```

【代码解析】

本实例关键功能是指示器的滑动。下面就是这个知识点的详细讲解。

在本实例中指示器的滑动有两种情况, 其一是用手拖曳指示器, 这时需要在 dragBegan: 方法中使用 frame 属性改变指示器的位置, 代码如下:

```
miniMeIndicator.frame = CGRectMake(touchPoint.x, touchPoint.y, miniMe.
frame.size.width/self.scrollView.zoomScale,
zoomScale, miniMe.frame.size.height/self.scrollView.zoomScale);
```

其二是滚动大视图,让指示器跟着移动,这时还是需要使用 frame 属性改变指示器的位置,代码如下:

```
miniMeIndicator.frame =CGRectMake(self.scrollView.contentOffset.x/_ratio/self.
scrollView.zoomScale,
self.scrollView.contentOffset.y/_ratio/self.scrollView.zoomScale,miniMe
Indicator.frame.size.width,miniMeIndicator.frame.size.height);
```

实例 26 具有放大镜的图像

【实例描述】

本实例实现的功能是在一个图像中添加上了一个放大镜。此放大镜可以放大在它下面的内容。运行效果如图 2.8 所示。

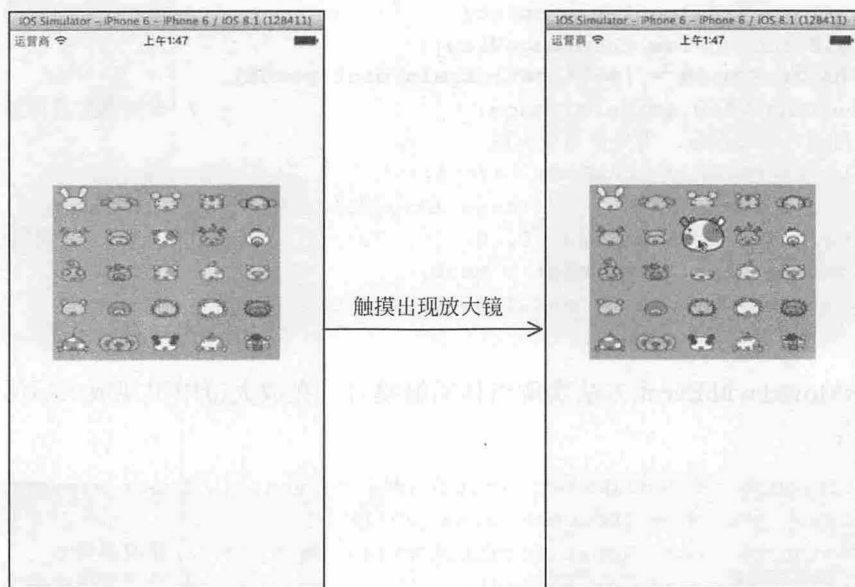


图 2.8 运行效果

【实现过程】

- (1) 创建一个项目,命名为“具有放大镜的图像”。
- (2) 添加图像 1.png、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIImageView 类的 MaskImageView 类。
- (4) 打开 MaskImageView.h 文件,编写代码,实现属性的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface MaskImageView : UIImageView
@property (nonatomic,strong) UIImageView *thumImageView; //属性
@end
```

(5) 打开 MaskImageView.m 文件, 编写代码, 实现放大镜的放大功能。使用的方法如表 2-7 所示。

表 2-7 MaskImageView.m文件中方法总结

方 法	功 能
touchesBegan:withEvent:	开始触摸
touchesMoved: withEvent:	移动触摸
touchesEnded: withEvent:	结束触摸
getImageInPoint:	获取某一点的图像

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中, touches Began:withEvent:方法实现开始触摸时, 在放大镜中出现放大后的图像。程序代码如下:

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];
    CGPoint point = [touch locationInView:self];           //获取触摸点
    //创建对设置图像视图, 用来保存放大后的图像
    thumImageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 70, 70)];
    thumImageView.center = point;                          //设置图像视图的中心位置
    [self addSubview:thumImageView];
    UIImage *image = [self getImageInPoint:point];
    thumImageView.image = image;                           //设置图像视图显示的图像
    //创建并设置图层, 用来作为放大镜
    CALayer *mask = [CALayer layer];
    mask.contents = (id)[[UIImage imageNamed:@"2.png"] CGImage];
    mask.frame = CGRectMake(0, 0, 70, 70);                 //设置框架
    thumImageView.layer.mask = mask;
    thumImageView.layer.masksToBounds = YES;
}
```

touchesMoved:withEvent:方法实现当移动触摸时, 在放大镜中出现放大后的图像。程序代码如下:

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];
    CGPoint point = [touch locationInView:self];           //获取触摸点
    thumImageView.center = point;                          //设置图像视图的中心位置
    UIImage *image = [self getImageInPoint:point];
    thumImageView.image = image;                           //设置图像视图显示的图像
    CALayer *mask = [CALayer layer];
    mask.contents = (id)[[UIImage imageNamed:@"2.png"] CGImage];
    mask.frame = CGRectMake(0, 0, 70, 70);                 //获取框架
    thumImageView.layer.mask = mask;
    thumImageView.layer.masksToBounds = YES;
    //判断 self.bounds 中是否包含 point
    if (!CGRectContainsPoint(self.bounds, point)) {
        [thumImageView removeFromSuperview];              //移除指定的视图
        thumImageView = nil;
    }
}
```

getImageInPoint:方法实现获取某一点的图像的功能。程序代码如下:

```
- (UIImage *)getImageInPoint:(CGPoint)point{
    UIImage* bigImage= [UIImage imageNamed:@"1.png"];           //创建图像对象
    //声明变量并为其赋值
    CGFloat x = point.x * bigImage.size.width/self.frame.size.width -35;
    CGFloat y = point.y * bigImage.size.height/self.frame.size.height -35;
    //实例化对象
    CGRect rect = CGRectMake(x, y, 70, 70);
    CGImageRef imageRef = bigImage.CGImage;
    CGImageRef subImageRef = CGImageCreateWithImageInRect(imageRef, rect);
    CGSize size;
    size.width = 70;
    size.height = 70;
    UIGraphicsBeginImageContext(size);           //创建一个基于位图的上下文
    CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
    CGContextDrawImage(context, rect, subImageRef); //绘制图像
    UIImage* smallImage = [UIImage imageWithCGImage:subImageRef];
    UIGraphicsEndImageContext();               //结束绘图, 并关闭绘图环境
    return smallImage;
}
```

(6) 打开 ViewController.h 文件, 编写代码, 实现头文件以及插座变量的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import "MaskImageView.h"
@interface ViewController : UIViewController{
    IBOutlet MaskImageView *maskView;           //声明插座变量
}
@end
```

(7) 打开 Main.storyboard 文件, 拖动 View 空白视图到设计界面中, 将 Class 设置为 MaskImageView, 将此视图和插座变量 maskView 关联。

(8) 打开 ViewController.m 文件, 编写代码, 实现对界面的初始化。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    UIImage *image = [self scaleFromImage:[UIImage imageNamed:@"1.png"]
    toSize:CGSizeMake(231, 178)];
    maskView.image = image;           //设置图像视图显示的图像
}
//缩放图像
- (UIImage *) scaleFromImage: (UIImage *) image toSize: (CGSize) size
{
    UIGraphicsBeginImageContext(size);           //创建一个基于位图的上下文
    [image drawInRect:CGRectMake(0, 0, size.width, size.height)];
    UIImage *newImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();               //结束绘图, 并关闭绘图环境
    return newImage;
}
```


【代码解析】

本实例关键功能是放大镜的显示以及图像的放大。下面依次讲解这两个知识点。

1. 放大镜的显示

在本实例中放大镜的显示需要 3 个步骤：

(1) 创建图层。在本实例中放大镜的显示是通过图层实现的，首先需要创建一个图层对象，使用的方法是 `layer`。代码如下：

```
CALayer *mask = [CALayer layer];
```

(2) 设置内容。图层的内容设置，需要使用 `contents` 属性实现，其语法形式如下：

```
@property(retain) id contents;
```

在本实例中，就是使用了 `contents` 属性，将图层的内容设置为图像 `2.png`。代码如下：

```
mask.contents = (id)[[UIImage imageNamed:@"2.png"] CGImage];
```

(3) 添加遮罩。最后需要为图像视图的图像添加遮罩（即创建的图层对象），使用的属性是 `mask`。代码如下：

```
thumbImageView.layer.mask = mask;
```

2. 图像的放大

要在放大镜中实现图像的放大也是需要 3 个步骤：

(1) 图像视图对象的创建。放大镜中放大图像的显示，需要创建一个图像视图对象，用来存放放大后的图像，代码如下：

```
thumbImageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 70, 70)];
```

(2) 获取图像。当放大镜到达某一位置后，就会放大此位置上的图像，这时需要使用 `getImageInPoint:` 方法，代码如下：

```
UIImage *image = [self getImageInPoint:point];
```

(3) 设置图像视图的图像。最后，需要将获取的图像设置为图像视图中要显示的图像，从而实现放大功能，代码如下：

```
thumbImageView.image = image;
```

实例 27 照片墙

【实例描述】

每一个照片都有着一段令人寻味的回忆。在生活中，人们希望将自己的回忆展示在家中的每一个角落，从而出现了照片墙。随着照片墙的日渐流行，手机应用上也开始仿照照片墙制作了一系列的应用在手机上的照片墙。该实例就为读者实现其中一种最为普遍的照片墙。当用户滚动屏幕时，照片墙就会滚动。运行效果如图 2.9 所示。

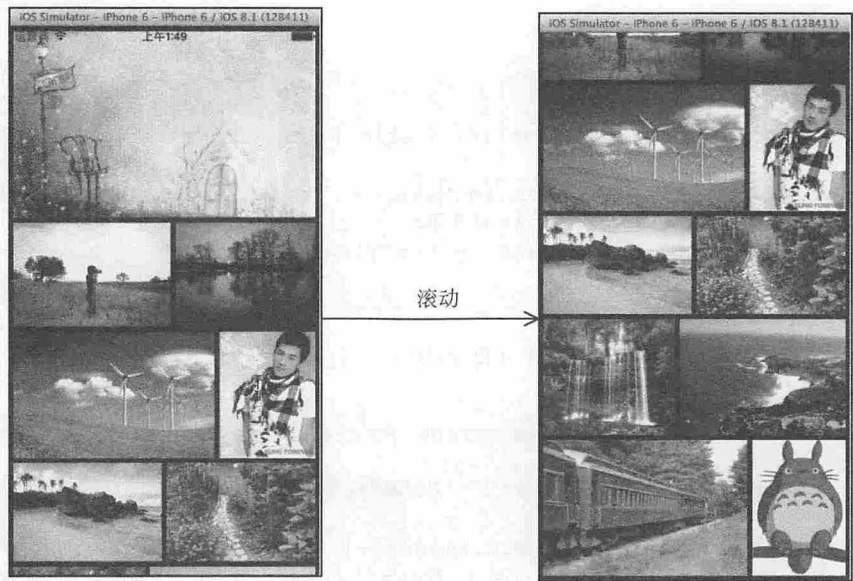


图 2.9 运行效果

【实现过程】

- (1) 创建一个项目，命名为“照片墙”。
- (2) 添加图像 0.jpg~10.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIViewController 类的 StyleViewController 类。
- (4) 打开 StyleViewController.h 文件，编写代码，实现属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface StyleViewController : UIViewController
//属性
@property (nonatomic, strong) NSArray *images;
@property (nonatomic, strong) UIScrollView *contentView;
- (id)initWithImages:(NSArray *)images; //初始化
- (void)placeImages; //图像的放置
- (CGSize)setFramesToImageViews:(NSArray *)imageViews toFitSize:(CGSize)
frameSize;
@end
```

- (5) 打开 StyleViewController.m 文件，编写代码，实现图像的放置。使用的方法如表 2-8 所示。

表 2-8 StyleViewController.m文件中方法总结

方 法	功 能
CGSizeResizeToHeight	获取尺寸
initWithImages:	初始化
loadView	加载视图
placeImages	图像的放置
setFramesToImageViews:toFitSize:	设置图像视图的框架

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。loadView 方法实现视

图的加载。程序代码如下：

```
- (void)loadView {
    [super loadView];
    self.contentView = [[UIScrollView alloc] initWithFrame:self.view.bounds];
    //实例化滚动视图

    self.contentView.autoresizingMask = UIViewAutoresizingFlexibleHeight|
    UIViewAutoresizingFlexibleWidth;
    [self.view addSubview:self.contentView];
    //添加视图对象
    [self placeImages];
}
```

placeImages 方法实现图像的放置（即添加）。程序代码如下：

```
- (void)placeImages {
    [self.contentView.subviews makeObjectsPerformSelector:@selector(remove
    FromSuperview)];
    NSMutableArray *imageViews = [NSMutableArray array];
    //遍历
    for (UIImage *image in self.images) {
        UIImageView *imageView = [[UIImageView alloc] initWithImage:image];
        [imageViews addObject:imageView];
        //添加对象
    }
    CGSize newSize = [self setFramesToImageViews:imageViews toFitSize:self.
    contentView.frame.size];
    self.contentView.contentSize = newSize;
    //设置可滚动区域的尺寸
    //遍历
    for (UIImageView *imageView in imageViews) {
        [self.contentView addSubview:imageView];
        //添加视图对象
    }
}
```

setFramesToImageViews:方法实现图像视图的框架设置。程序代码如下：

```
- (CGSize)setFramesToImageViews:(NSArray *)imageViews toFitSize:(CGSize)
frameSize {
    int N = imageViews.count;
    //获取数组 imageViews 中的个数
    CGRect newFrames[N];
    float ideal_height = MAX(frameSize.height, frameSize.width) / 4;
    float seq[N];
    float total_width = 0;
    //循环
    for (int i = 0; i < imageViews.count; i++) {
        UIImage *image = [[imageViews objectAtIndex:i] image]; //实例化对象
        CGSize newSize = CGSizeResizeToHeight(image.size, ideal_height);
        newFrames[i] = (CGRect) {{0, 0}, newSize}; //为 newFrames 中的元素赋值
        seq[i] = newSize.width;
        total_width += seq[i];
        //获取宽度
    }
    int K = (int)roundf(total_width / frameSize.width);
    //声明变量，用来设置数组中的元素

    float M[N][K];
    float D[N][K];
    //循环，将元素设置为 0
    for (int i = 0; i < N; i++)
        for (int j = 0; j < K; j++)
            D[i][j] = 0;
    for (int i = 0; i < K; i++)
        M[0][i] = seq[0];
    //设置 0 行 i 列的值
```

```

for (int i = 0; i < N; i++)
    M[i][0] = seq[i] + (i ? M[i-1][0] : 0);           //设置 i 行 0 列的值
float cost;
//循环
for (int i = 1; i < N; i++) {
    for (int j = 1; j < K; j++) {
        M[i][j] = INT_MAX;
        for (int k = 0; k < i; k++) {
            cost = MAX(M[k][j-1], M[i][0]-M[k][0]); //获取最大值
            //判断 M[i][j] 是否大于 cost
            if (M[i][j] > cost) {
                M[i][j] = cost;
                D[i][j] = k;
            }
        }
    }
}
int k1 = K-1;
int n1 = N-1;
int ranges[N][2];
while (k1 >= 0) {
    ranges[k1][0] = D[n1][k1]+1;
    ranges[k1][1] = n1;
    n1 = D[n1][k1];
    k1--;
}
ranges[0][0] = 0;
//为变量赋值
float cellDistance = 5;
float heightOffset = cellDistance, widthOffset;
float frameWidth;
for (int i = 0; i < K; i++) {
    float rowWidth = 0;
    frameWidth = frameSize.width - ((ranges[i][1] - ranges[i][0]) + 2)*
    cellDistance; //框架的宽度
    for (int j = ranges[i][0]; j <= ranges[i][1]; j++) {
        rowWidth += newFrames[j].size.width; //获取宽度
    }
    float ratio = frameWidth / rowWidth;
    widthOffset = 0;
    for (int j = ranges[i][0]; j <= ranges[i][1]; j++) {
        newFrames[j].size.width *= ratio; //设置宽度
        newFrames[j].size.height *= ratio; //设置高度
        newFrames[j].origin.x = widthOffset + (j - (ranges[i][0]) + 1)*
        cellDistance;
        newFrames[j].origin.y = heightOffset;
        widthOffset += newFrames[j].size.width;
    }
    heightOffset += newFrames[ranges[i][0]].size.height + cellDistance;
    //获取高度的偏移量
}
//添加图像视图对象
for (int i = 0; i < N; i++) {
    UIImageView *imgView = imageView[i];
    imgView.frame = newFrames[i]; //设置框架
    [self.contentView addSubview:imgView];
}
return CGSizeMake(frameSize.width, heightOffset);
}

```

(6) 打开 ViewController.h 文件，编写代码，实现头文件以及对象的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "StyleViewController.h"
@interface ViewController : UIViewController{
    StyleViewController *styleView;
}
@end
```

(7) 打开 ViewController.m 文件，编写代码，实现照片墙的显示。程序代码如下：

```
-(void)viewDidLoad{
    NSMutableArray *images = [NSMutableArray array];
    //循环
    for (int i = 0; i <= 10; i++) {
        NSString *fileName = [NSString stringWithFormat:@"%i.jpg", i];
        [images addObject:[UIImage imageNamed:fileName]]; //添加对象
    }
    for (int i = 0; i <= 10; i++) {
        [images exchangeObjectAtIndex:i withObjectAtIndex:arc4random_uniform(10)]; //交换对象
    }
    styleView=[[StyleViewController alloc] initWithImages:images]; //实例化对象
    [self.view addSubview:styleView.view];
}
```

【代码解析】

本实例关键功能是照片墙的风格。下面就是这个知识点的详细讲解。

在此实例中，照片墙的每一次运行风格是不一样的，如果想要实现照片墙的风格，首先，需要随机地出现图像，此时需要使用 exchangeObjectAtIndex:withObjectAtIndex:方法，它可以实现两个对象交换。其语法形式如下：

```
-(void)exchangeObjectAtIndex:(NSUInteger)idx1 withObjectAtIndex:(NSUInteger)idx2;
```

其中，(NSUInteger)idx1 表示对象的索引，(NSUInteger)idx2 表示对象的索引。在本实例中的代码如下：

```
[images exchangeObjectAtIndex:i withObjectAtIndex:arc4random_uniform(10)];
```

其中，i 表示对象的索引，arc4random_uniform(10)表示对象的索引。其中，需要改变图像出现的位置以及大小，在本实例中自定义了一个 setFrameToImageViews:toFitSize:方法，它的功能是获取图像的宽度以及高度。

实例 28 图像对比

【实例描述】

本实例实现的功能是一个图像对比的功能。左右两边分别是两张不一样的图片。在中间会有一个红色的线，此线就是图片对比的分隔线。在分隔线上有一个滑块，用于让用户

向左或者向右去滑动。运行效果如图 2.10 所示。



图 2.10 运行效果

【实现过程】

- (1) 创建一个项目，命名为“图像对比”。
- (2) 添加图像 1.png、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 Slider 类。
- (4) 打开 Slider.m 文件，编写代码，实现分隔线的初始化以及绘制。程序代码如下：

```
//初始化
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor clearColor]; //设置背景颜色
    }
    return self;
}

//绘制分隔线
- (void)drawRect:(CGRect)rect
{
    [super drawRect:rect];
    CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
    CGContextBeginPath(context);
    //绘制线段
    CGContextMoveToPoint(context, rect.size.width/2, 0);
    CGContextAddLineToPoint(context, rect.size.width/2, rect.size.height);
    CGContextSetLineWidth(context, 2.0f); //设置线宽
    CGContextSetStrokeColorWithColor(context, [UIColor redColor].CGColor); //设置线的颜色
    CGContextStrokePath(context);
}
```

- (5) 创建一个基于 UIView 类的 Compare 类。

(6) 打开 Compare.h 文件，编写代码，实现属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface Compare : UIView
//属性
@property (nonatomic,strong) UIView *sliderView;
@property (nonatomic,assign) float sliderPosition;
@property (nonatomic,assign) BOOL isSliderTouched;
@property (nonatomic,strong) UIImageView *leftImageView;
@property (nonatomic,strong) UIImageView *rightImageView;
//方法
- (void)setSliderPosition:(float)sliderPosition animated:(BOOL)animated;
- (id)initWithLeftSideImage:(UIImage *)leftImage rightSideImage:(UIImage *)rightImage;
@end
```

(7) 打开 Compare.m 文件，编写代码，实现初始化以及移动分隔线进行图像的对比。使用的方法如表 2-9 所示。

表 2-9 Compare.m 文件中方法总结

方 法	功 能
initWithLeftSideImage:rightSideImage:	初始化
setSliderView:	设置分隔线视图
setSliderPosition:animated:	设置分隔线的位置以及动画
setSliderPosition:	设置分隔线的位置
touchesBegan:withEvent:	开始触摸
touchesMoved:withEvent:	移动触摸
touchesEnded:withEvent:	结束触摸

其中，要实现初始化，需要使用 initWithLeftSideImage:rightSideImage: 和 setSliderView:、setSliderPosition: 方法。initWithLeftSideImage:rightSideImage: 方法实现对两个对比的图像以及分隔线进行初始化。程序代码如下：

```
- (id)initWithLeftSideImage:(UIImage *)leftImage rightSideImage:(UIImage *)rightImage
{
    if (self = [super initWithFrame:CGRectMake(0, 0, leftImage.size.width, leftImage.size.height)]) {
        //左边图像视图对象的初始化以及设置
        _leftImageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, leftImage.size.width, leftImage.size.height)]; //实例化图像视图对象
        _leftImageView.contentMode = UIViewContentModeLeft;
        _leftImageView.clipsToBounds = YES;
        _leftImageView.image = leftImage; //设置图像视图显示的图像
        //右边图像视图对象的初始化以及设置
        _rightImageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, leftImage.size.width, leftImage.size.height)];
        _rightImageView.contentMode = UIViewContentModeRight;
        _rightImageView.image = rightImage; //设置图像视图显示的图像
        _rightImageView.clipsToBounds = YES;
        //设置图像视图对象的中心位置
        _leftImageView.center = self.center;
        _rightImageView.center = self.center;
        [self addSubview:_leftImageView];
```

```

        [self addSubview:_rightImageView];
        //对分隔线位置的设置
        self.sliderPosition = leftImage.size.width/2;
    }
    return self;
}

```

setSliderView:方法实现对分隔线的设置。程序代码如下:

```

- (void)setSliderView:(UIView *)sliderView
{
    [_sliderView removeFromSuperview];           //移除指定的视图对象
    _sliderView = sliderView;
    _sliderView.center = self.center;             //设置中心位置
    [self addSubview:_sliderView];
}

```

setSliderPosition:方法实现定义分隔线位置的设置。程序代码如下:

```

- (void)setSliderPosition:(float)sliderPosition
{
    if ((sliderPosition < self.frame.size.width) && (sliderPosition > self.
        bounds.origin.x)) {
        _sliderPosition = sliderPosition;
        if (self.sliderView) {
            self.sliderView.center = CGPointMake(sliderPosition, self.slider
                View.center.y);           //设置中心位置
        }
        CGRect leftImageRect = self.leftImageView.frame; //获取框架
        leftImageRect.size.width = sliderPosition;
        self.leftImageView.frame = leftImageRect;         //设置框架
        CGRect rightImageRect = self.rightImageView.frame;
        rightImageRect.origin.x = sliderPosition;
        rightImageRect.size.width = self.frame.size.width - sliderPosition;
                                                                    //设置宽度
        self.rightImageView.frame = rightImageRect;
    }
}

```

移动分隔线进行图像的对比需要使用 setSliderPosition:animated:、setSliderPosition:、touchesBegan:withEvent:、touchesMoved:withEvent:、touchesEnded:withEvent:。其中, setSliderPosition:animated:方法实现对分隔线的位置以及动画进行设置。程序代码如下:

```

- (void)setSliderPosition:(float)sliderPosition animated:(BOOL)animated
{
    if (animated) {
        //分隔线移动的动画效果
        [UIView animateWithDuration:1.0 delay:0.0 options:UIViewAnimation
            OptionBeginFromCurrentState animations:^(
                self.sliderPosition = sliderPosition;           //设置位置
            ) completion:nil];
    } else {
        self.sliderPosition = sliderPosition;
    }
}

```

touchesBegan:withEvent:、touchesMoved:withEvent:、touchesEnded:withEvent:方法实现的是手势的触摸功能。程序代码如下:

```

//开始触摸
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint currentPoint = [touch locationInView:self.sliderView];
    //获取触摸点

    //判断 currentPoint 是否包含在 self.sliderView.bounds 中
    if (CGRectContainsPoint(self.sliderView.bounds, currentPoint)) {
        self.isSliderTouched = YES;
    }
}

//移动触摸
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (self.isSliderTouched) {
        UITouch *touch = [touches anyObject];
        CGPoint currentPoint = [touch locationInView:self]; //获取触摸点
        if (currentPoint.x >= self.sliderView.bounds.size.width/2 &&
            currentPoint.x <= self.bounds.size.width - self.sliderView.bounds.size.width/2) {
            self.sliderPosition = currentPoint.x;
            //设置位置
        }
    }
}

//结束触摸
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (self.isSliderTouched == NO) {
        UITouch *touch = [touches anyObject];
        CGPoint currentPoint = [touch locationInView:self]; //获取触摸点
        [self setSliderPosition:currentPoint.x animated:YES];
    }
    self.isSliderTouched = NO;
}

```

(8) 打开 ViewController.h 文件，编写代码，实现头文件的声明。

(9) 打开 ViewController.m 文件，编写代码，实现对界面的初始化。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    Compare *before = [[Compare alloc] initWithLeftSideImage:[UIImage
        imageNamed:@"2.png"] rightSideImage:[UIImage imageNamed:@"1.png"]];
    //实例化对象

    UIImageView *slider = [[UIImageView alloc] initWithImage:[UIImage
        imageNamed:@"3.png"]];
    //创建分隔线
    before.sliderView = [[Slider alloc] initWithFrame:CGRectMake(0, 0,
        slider.frame.size.width, before.frame.size.height)];
    [before.sliderView addSubview:slider];
    before.sliderView.contentMode = UIViewContentModeCenter; //设置显示模式
    slider.center = CGPointMake(slider.center.x, before.sliderView.

```

```

        center.y);
        [self.view addSubview:before];
        before.center = self.view.center;           //设置中心位置
    }

```

【代码解析】

本实例关键功能是图像的对象。下面就是这个知识点的详细讲解。

在本实例中图像的对比工作都是由分隔线的位置决定的。要实现图像的对比,首先需要创建并设置两个对比的图像,代码如下:

```

_leftImageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0,
leftImage.size.width, leftImage.size.height)];
_leftImageView.contentMode = UIViewContentModeLeft;
.....

_rightImageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0,
leftImage.size.width, leftImage.size.height)];
_rightImageView.contentMode = UIViewContentModeRight;
.....

[self addSubview:_leftImageView];
[self addSubview:_rightImageView];

```

其次,需要对分隔线的位置进行设置,从而实现图像的对比。代码如下:

```

- (void)setSliderPosition:(float)sliderPosition
{
    if ((sliderPosition < self.frame.size.width) && (sliderPosition >
self.bounds.origin.x)) {
        _sliderPosition = sliderPosition;
        .....
        //左边图像的框架
        CGRect leftImageRect = self.leftImageView.frame;
        leftImageRect.size.width = sliderPosition;
        self.leftImageView.frame = leftImageRect;
        //右边图像的框架
        CGRect rightImageRect = self.rightImageView.frame;
        rightImageRect.origin.x = sliderPosition;
        rightImageRect.size.width = self.frame.size.width - sliderPosition;
        self.rightImageView.frame = rightImageRect;
    }
}

```

实例 29 刮刮卡

【实例描述】

本实例实现的功能是一个刮刮卡的功能,当用户用于在显示有刮奖区的区域来回滑动,就会出现被刮奖区覆盖的区域。运行效果如图 2.11 所示。

【实现过程】

- (1) 创建一个项目,命名为“刮刮卡”。
- (2) 添加图像 1.jpg、2.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 View 类。

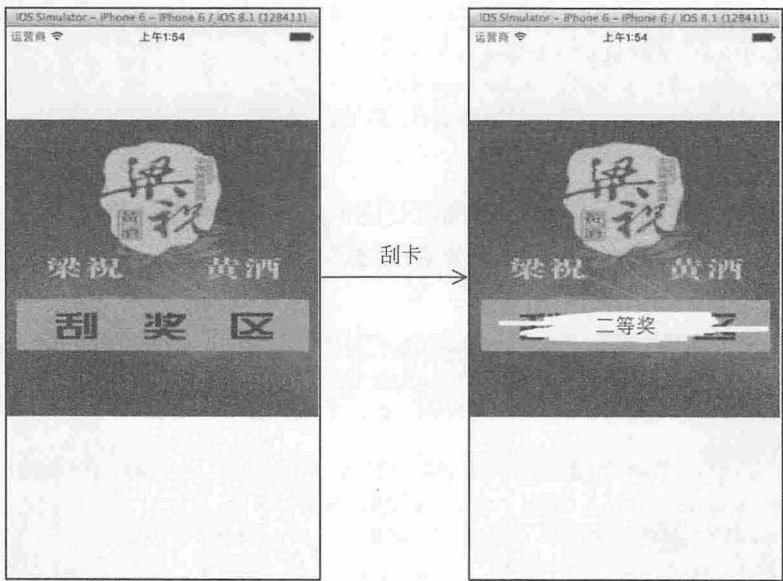


图 2.11 运行效果

(4) 打开 View.h 文件，编写代码，实现实例变量、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface View : UIView{
    //实例变量
    CGPoint previousTouchLocation;
    CGPoint currentTouchLocation;
    CGImageRef hideImage;
    CGImageRef scratchImage;
    CGContextRef contextMask;
}
//属性
@property (nonatomic, assign) float percentAccomplishment;
@property (nonatomic, assign) float sizeBrush;
@property (nonatomic, strong) UIView *hideView;
- (void)setHideView:(UIView *)hideView; //设置隐藏的视图
@end
```

(5) 打开 View.m 文件，编写代码，实现刮刮卡的功能。使用的方法如表 2-10 所示。

表 2-10 View.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
drawRect:	将图像绘制到矩形区域中
setHideView:	设置隐藏的视图
scratchTheViewFrom:	绘制刮痕
touchesBegan:withEvent:	开始触摸
touchesMoved:withEvent:	移动触摸
touchesEnded:withEvent:	结束触摸
touchesCancelled:withEvent:	隐藏触摸

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，初始化的设置，

需要使用 `initWithFrame:`、`drawRect:`、`setHideView:`方法。`drawRect:`方法实现将图像绘制到指定的矩形区域中。程序代码如下:

```
- (void)drawRect:(CGRect)rect
{
    [super drawRect:rect];
    UIImage *imageToDraw = [UIImage imageWithCGImage:scratchImage];
    [imageToDraw drawInRect:CGRectMake(0.0, 0.0, self.frame.size.width,
    self.frame.size.height)]; //绘制
}
```

`setHideView:`方法实现设置隐藏的视图。程序代码如下:

```
- (void)setHideView:(UIView *)hideView
{
    CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceGray();
    //创建色彩空间
    int bitmapByteCount;
    int bitmapBytesPerRow;
    float scale = [UIScreen mainScreen].scale; //获取缩放因子
    //创建位图的上下文,并设置缩放因子
    UIGraphicsBeginImageContextWithOptions(hideView.bounds.size, NO, 0);
    [hideView.layer renderInContext:UIGraphicsGetCurrentContext()];
    hideView.layer.contentsScale = scale;
    hideImage = UIGraphicsGetImageFromCurrentImageContext().CGImage;
    UIGraphicsEndImageContext(); //结束绘图,并关闭绘图环境
    size_t imageWidth = CGImageGetWidth(hideImage); //获取宽度
    size_t imageHeight = CGImageGetHeight(hideImage); //获取高度
    bitmapBytesPerRow = (imageWidth * 4);
    bitmapByteCount = (bitmapBytesPerRow * imageHeight);
    CFMutableDataRef pixels = CFDataCreateMutable(NULL, imageWidth *
    imageHeight);
    contextMask = CGBitmapContextCreate(CFDataGetMutableBytePtr(pixels),
    imageWidth, imageHeight, 8, imageWidth, colorspace, kCGImageAlphaNone);
    //创建位图上下文
    CGDataProviderRef dataProvider = CGDataProviderCreateWithCFData(pixels);
    CGContextSetFillColorWithColor(contextMask, [UIColor blackColor].CGColor);
    //设置填充颜色
    CGContextFillRect(contextMask, self.frame); //绘制
    CGContextSetStrokeColorWithColor(contextMask, [UIColor whiteColor].
    CGColor);
    CGContextSetLineWidth(contextMask, _sizeBrush); //设置线宽
    CGContextSetLineCap(contextMask, kCGLineCapRound);
    CGImageRef mask = CGImageMaskCreate(imageWidth, imageHeight, 8, 8,
    imageWidth, dataProvider, nil, NO);
    scratchImage = CGImageCreateWithMask(hideImage, mask);
    CGImageRelease(mask);
    CGColorSpaceRelease(colorspace); //释放色彩空间
}
```

如果想要实现刮卡的功能,需要使用 `scratchTheViewFrom:`、`touchesBegan:withEvent:`、`touchesMoved:withEvent:`、`touchesEnded:withEvent:`、`touchesCancelled:withEvent:`。`scratchTheViewFrom:to:`方法实现对刮痕的绘制。程序代码如下:

```
- (void)scratchTheViewFrom:(CGPoint)startPoint to:(CGPoint)endPoint
{
    float scale = [UIScreen mainScreen].scale;
    CGContextMoveToPoint(contextMask, startPoint.x * scale, (self.frame.
```

```

size.height - startPoint.y) * scale); //设置开始点
CGContextAddLineToPoint(contextMask, endPoint.x * scale, (self.frame.
size.height - endPoint.y) * scale); //设置结束点
CGContextStrokePath(contextMask);
[self setNeedsDisplay];
}

```

touchesBegan:withEvent:、touchesMoved:withEvent:、touchesEnded:withEvent:、touchesCancelled:withEvent:方法实现触摸的功能（即刮卡的功能）。程序代码如下：

```

//开始触摸
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    [super touchesBegan:touches withEvent:event];
    UITouch *touch = [[event touchesForView:self] anyObject];
    currentTouchLocation = [touch locationInView:self]; //获取触摸点
}
//移动触摸
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    [super touchesMoved:touches withEvent:event];
    UITouch *touch = [[event touchesForView:self] anyObject];
    //判断 previousTouchLocation 与 CGPointZero 是否相同
    if (!CGPointEqualToPoint(previousTouchLocation, CGPointZero))
    {
        currentTouchLocation = [touch locationInView:self]; //获取触摸点
    }
    previousTouchLocation = [touch previousLocationInView:self];
    //获取前一个触摸点
    [self scratchTheViewFrom:previousTouchLocation to:currentTouchLocation];
}
//结束触摸
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    [super touchesEnded:touches withEvent:event];
    UITouch *touch = [[event touchesForView:self] anyObject];
    //判断 previousTouchLocation 与 CGPointZero 是否相同
    if (!CGPointEqualToPoint(previousTouchLocation, CGPointZero))
    {
        previousTouchLocation = [touch previousLocationInView:self];
        //获取触摸点
        [self scratchTheViewFrom:previousTouchLocation to:currentTouchLocation];
    }
}
//取消触摸
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
    [super touchesCancelled:touches withEvent:event];
}

```

(6) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "View.h"
@interface ViewController : UIViewController{
    //插座变量

```

```

IBOutlet UILabel *dollarsAmount;
IBOutlet View *scratchView;
}
@end

```

(7) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 2.12 所示。

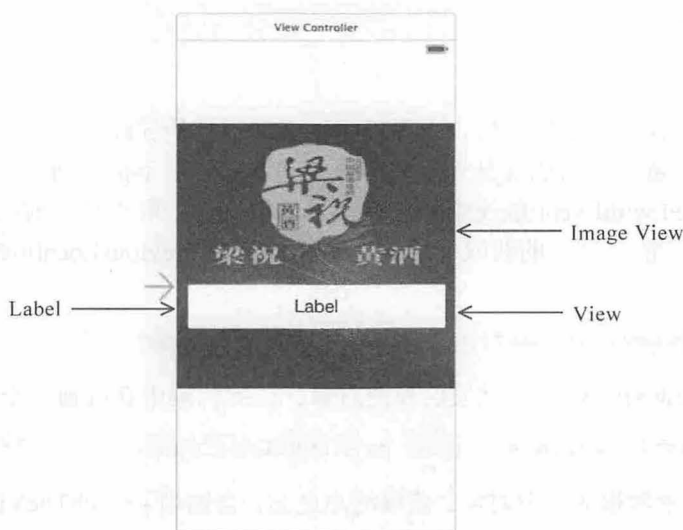


图 2.12 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-11 所示。

表 2-11 视图、控件设置

视图、控件	属性设置	其他
Image View	Image View: 1.jpg	
View		Class: View 与插座变量scratchView关联
Label	Font: System 21.0 Alignment: 居中	与插座变量dollarsAmount关联

(8) 打开 ViewController.m 文件, 编写代码, 实现对界面的设置。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [scratchView setSizeBrush:10.0];
    UIImageView *ball = [[UIImageView alloc] initWithFrame:CGRectMake(0.0,
    0.0, 303.0, 60.0)]; //实例化对象
    [ball setImage:[UIImage imageNamed:@"2.png"]]; //设置图像视图显示的图像
    [scratchView setHiddenView:ball];
    int i=arc4random()%4;
    switch (i) {
        case 0:
            dollarsAmount.text=@"一等奖"; //设置文本内容
            break;
        case 1:

```

```

        dollarsAmount.text=@"二等奖";           //设置文本内容
        break;
    case 2:
        dollarsAmount.text=@"三等奖";           //设置文本内容
        break;
    default:
        dollarsAmount.text=@"谢谢惠顾";         //设置文本内容
        break;
    }
}

```

【代码解析】

本实例关键功能是划痕的效果。下面就是这个知识点的详细讲解。

在刮刮卡中最重要的功能就是划痕的实现。当用户使用手指在刮奖区移动时，会自动调用 `touchesMoved:withEvent:` 触摸事件。在此事件中，会获取用户当前的点以及当前点的前一个点，其中，前一个点的获取使用的是 `UITouch` 的 `previousLocationInView:` 方法，其语法形式如下：

```
- (CGPoint)previousLocationInView:(UIView *)view;
```

其中，`(UIView *)view` 表示触摸的视图对象。在此代码中获取前一个点的代码如下：

```
previousTouchLocation = [touch previousLocationInView:self];
```

其中，`self` 表示触摸的视图对象。获取两点之后，会调用 `scratchTheViewFrom:to:` 方法，代码如下：

```
[self scratchTheViewFrom:previousTouchLocation to:currentTouchLocation];
```

此方法的功能是对划痕进行绘制。

实例 30 GIF 图像的显示

【实例描述】

在 iOS 中显示图像，千万不要以为只可以显示 `png`、`jpg` 等静态的图像，它还可以显示类似于 `gif` 格式的图像。本实例就为读者实现一种在 iOS 应用程序中显示 `gif` 格式的图像的功能。运行效果如图 2.13 所示。

【实现过程】

- (1) 创建一个项目，命名为“GIF 图像的显示”。
- (2) 添加图像 `1.gif` 到创建项目的 `Supporting Files` 文件夹中。
- (3) 打开 `ViewController.m` 文件，编写代码，实现 `Gif` 图像的显示功能。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view,
    typically from a nib.
    NSString *path = [[NSBundle mainBundle] path

```

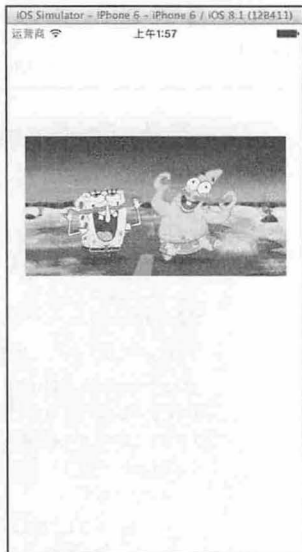


图 2.13 运行效果

```

ForResource:@"1" ofType:
    @"gif"];
NSData *gifData = [NSData dataWithContentsOfFile:path];
UIWebView *webView = [[UIWebView alloc] initWithFrame:CGRectMake(20, 120,
280, 150)];
webView.backgroundColor = [UIColor redColor];           //设置背景颜色
webView.scalesPageToFit = YES;                           //自动缩放
[webView loadData:gifData MIMEType:@"image/gif" textEncodingName:nil
baseURL:nil];      //加载数据
[self.view addSubview:webView];
}

```

【代码解析】

本实例关键功能是 GIF 图像的显示。下面就是这个知识点的详细讲解。

UIWebView 的 loadData:MIMEType:textEncodingName:baseURL:方法可以实现转载本地页面或者外部传来的 NSData。程序代码如下:

```

- (void)loadData:(NSData *)data MIMEType:(NSString *)MIMEType textEncoding
Name:(NSString *)encodingName baseURL:(NSURL *)baseURL;

```

其中, 参数说明如下:

- (NSData *)data 表示数据对象;
- (NSString *)MIMEType 表示 MIME 类型;
- (NSString *)encodingName 表示编码类型;
- (NSURL *)baseURL 表示 URL。

在此代码中就是用 loadData:MIMEType:textEncodingName:baseURL:方法实现了 GIF 图像的显示。代码如下:

```

[webView loadData:gifData MIMEType:@"image/gif" textEncodingName:nil
baseURL:nil];

```

其中, 参数说明如下:

- gifDat 表示数据对象;
- @"image/gif"表示 MIME 类型;
- 第 1 个 nil 表示编码类型;
- 第 2 个 nil 表示 URL。

实例 31 评分控件

【实例描述】

在购买商品后, 很多的卖家都会让买家进行评论并且评分, 这也是吸引买家的一种手段。本实例就实现一个评分的功能。当用户滑动滑块控件中的滑块后, 就会实现评分的功能。运行效果如图 2.14 所示。

【实现过程】

- (1) 创建一个项目, 命名为“评分控件”。



图 2.14 运行效果

- (2) 创建一个基于 UIView 类的 View 类。
- (3) 打开 View.h 文件，编写代码，实现宏定义以及属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#define th M_PI/180
@interface View : UIView
@property(nonatomic) CGFloat radius;
@property(nonatomic) CGFloat value; // 范围 0 到 1
@property(nonatomic,strong) UIColor *startColor;
@property(nonatomic,strong) UIColor *boundsColor;
@end
```

- (4) 打开 View.m 文件，编写代码，实现绘制五角星的功能。使用的方法如表 2-12 所示。

表 2-12 View.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
setFrame:	设置框架
initWithCoder:	初始化实例化对象
drawRect:	绘制五角星
setValue:	设置值

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，initWithFrame: 方法实现对视图的初始化设置。程序代码如下：

```
-(id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        CGFloat x = frame.size.width / 2;
```

```

CGFloat y = frame.size.height / 2;
self.radius = x < y ? x:y;
self.value = 1;
self.startColor = [UIColor yellowColor];           //设置开始颜色
self.backgroundColor = [UIColor clearColor];       //设置背景颜色
self.boundsColor = [UIColor blackColor];           //设置边界颜色
}
return self;
}

```

drawRect:方法实现了五角星的绘制。程序代码如下:

```

- (void)drawRect:(CGRect)rect
{
    CGFloat centerX = rect.size.width / 2;
    CGFloat centerY = rect.size.height / 2;
    CGFloat r0 = self.radius * sin(18 * th)/cos(36 * th); /*计算小圆半径 r0 */
    CGFloat x1[5]={0},y1[5]={0},x2[5]={0},y2[5]={0};
    //计算坐标
    for (int i = 0; i < 5; i ++)
    {
        x1[i] = centerX + self.radius * cos((90 + i * 72) * th);
        //计算出大圆上的五个平均分布点的坐标
        y1[i]=centerY - self.radius * sin((90 + i * 72) * th);
        x2[i]=centerX + r0 * cos((54 + i * 72) * th);
        //计算出小圆上的五个平均分布点的坐标
        y2[i]=centerY - r0 * sin((54 + i * 72) * th);
    }
    CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
    CGMutablePathRef startPath = CGPathCreateMutable();
    CGPathMoveToPoint(startPath, NULL, x1[0], y1[0]); //设置开始点
    //绘制圆
    for (int i = 1; i < 5; i ++) {
        CGPathAddLineToPoint(startPath, NULL, x2[i], y2[i]); //设置结束点
        CGPathAddLineToPoint(startPath, NULL, x1[i], y1[i]);
    }
    CGPathAddLineToPoint(startPath, NULL, x2[0], y2[0]);
    CGPathCloseSubpath(startPath); //关闭子路径
    CGContextAddPath(context, startPath); //添加路径
    //设置颜色
    CGContextSetFillColorWithColor(context, self.startColor.CGColor);
    CGContextSetStrokeColorWithColor(context, self.boundsColor.CGColor);
    //设置线的颜色
    CGContextStrokePath(context);
    CGRect range = CGRectMake(x1[1], 0, (x1[4] - x1[1]) * self.value, y1[2]);
    CGContextAddPath(context, startPath); //添加路径
    CGContextClip(context);
    CGContextFillRect(context, range); //填充
    CFRelease(startPath);
}

```

setValue:方法实现对值的绘制,此值是用来对五角星填充的关键。程序代码如下:

```

- (void) setValue:(CGFloat)value
{
    //判断 value 值是否为 0
    if (value < 0) {
        _value = 0;
    }
}

```

```
    }else if(value > 1){                                //判断 value 值是否大于 1
        _value = 1;
    }else{
        _value = value;
    }
    [self setNeedsDisplay];                            //重新绘制
}
```

(5) 创建一个基于 UIView 类的 RateView 类。

(6) 打开 RateView.h 文件，编写代码，实现头文件、属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "View.h"
@interface RateView : UIView
//属性
@property(nonaatomic) float rate;
@property(nonaatomic) int maxRate;
.....
@property(nonaatomic,strong) NSMutableArray *rateViews;
//方法
-(void) baseInit;
-(void) refresh;
@end
```

(7) 打开 RateView.m 文件，编写代码，实现评分的功能。使用的方法如表 2-13 所示。

表 2-13 RateView.m 文件中方法总结

方 法	功 能
baseInit	初始化设置
refresh	刷新
initWithFrame:	评分控件的初始化
initWithCoder:	初始化实例化对象
layoutSubviews	布局
setMaxRate:	设置最大的评分等级
setRate:	设置评分等级

baseInit 方法实现对一些默认的设置进行初始化。程序代码如下：

```
-(void)baseInit
{
    self.rateViews = [[NSMutableArray alloc] init];        //创建可变数组
    //对属性进行设置
    self.rate = 0;
    self.maxRate = 5;
    self.leftMargin = 0;
    self.rightMargin = 0;
    self.midMargin = 8;
    self.minImageSize = CGRectMake(0, 0, 16, 16);
}
```

refresh 方法实现对五角星视图值的设置。程序代码如下：

```

-(void) refresh
{
    int size = self.rateViews.count;           //获取数组中元素的个数并赋值给 size
    for (int i = 0; i < size; i++) {
        View *view = [self.rateViews objectAtIndex:i];
        //判断 self.rate 的值是否大于 i+1
        if (self.rate >= i + 1) {
            view.value = 1;
        } else if (self.rate > i) {             //判断 self.rate 的值是否大于 i
            view.value = self.rate - i;
        } else {
            view.value = 0;
        }
    }
}

```

layoutSubviews 方法实现对五角星视图的重新布局。程序代码如下:

```

-(void) layoutSubviews
{
    [super layoutSubviews];
    float desireImageViewWidth = (self.frame.size.width - self.leftMargin
    -self.rightMargin - self.midMargin * (self.maxRate - 1)) / self.maxRate;
    float imageWidth = MAX(self.minImageSize.size.width, desireImage
    ViewWidth);                               //获取最大的宽度
    float imageHeight = MAX(self.frame.size.height, self.minImageSize.
    size.height);                             //获取最大的高度
    int size = 5;
    UIImageView *view = NULL;
    //循环
    for (int i = 0; i < size; i++) {
        //判断 i 是否小于 3
        if ( i < 3) {
            view = [self.rateViews objectAtIndex:i];
            view.frame = CGRectMake(self.leftMargin + (self.midMargin + image
            Width)* i , 50*i, imageWidth, imageHeight);           //设置框架
        } else {
            view = [self.rateViews objectAtIndex:i];
            view.frame = CGRectMake(self.leftMargin + (self.midMargin + image
            Width)* i , 50*(size-i-1), imageWidth, imageHeight); //设置框架
        }
    }
}

```

setMaxRate:方法实现对评分中最大等级的设置。程序代码如下:

```

-(void) setMaxRate:(int)maxRate
{
    _maxRate = maxRate;
    int size = self.rateViews.count;           //获取数组中元素的个数赋值给 size
    View *view = NULL;
    //判断 size 是否大于 maxRate
    if (size > maxRate) {
        for (int i = size - 1; size >= maxRate; i--) {

```

```

        view = (View*) [self.rateViews objectAtIndex:i];
        [view removeFromSuperview]; //移除指定的视图对象
        [self.rateViews removeObjectAtIndex:i]; //移除对象
    }
    [self setNeedsLayout];
    [self refresh]; //调用 refresh 方法实现刷新
} else {
    for (int i = size; i < maxRate; i++) {
        view = [[View alloc] initWithFrame:CGRectMake(0, 0, 10 , 10)];
        [self.rateViews addObject:view]; //添加对象
        [self addSubview:view];
    }
    [self setNeedsLayout];
    [self refresh]; //调用 refresh 方法实现刷新
}
}

```

(8) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量以及动作的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "RateView.h"
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet UISlider *slider;
    IBOutlet RateView *rateView;
}
- (IBAction)valueChange:(id)sender;
@end

```

(9) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 2.15 所示。

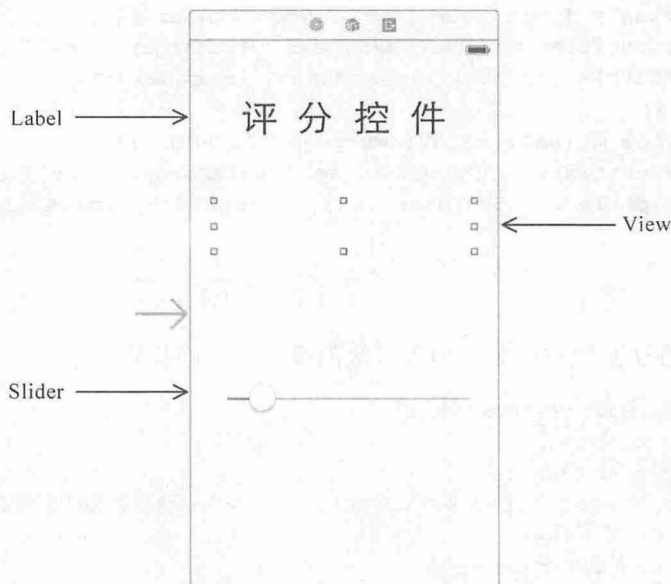


图 2.15 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-14 所示。

表 2-14 视图、控件设置

视图、控件	属性设置	其他
Label	Text: 评分控件 Font: System 38.0 Alignment: 居中	
View		Class: RateView 与插座变量rateView关联
Slider		与插座变量slider关联 与动作valueChange:关联

(10) 打开 ViewController.m 文件, 编写代码, 实现初始化以及拖动滑块进行评分的功能。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    rateView.rate = 3.8;
    slider.value = 3.8; //设置滑块控件的值
}
- (IBAction)valueChange:(id)sender {
    rateView.rate = slider.value;
}
```

【代码解析】

本实例关键功能是评分控件中评分的实现。下面就是这个知识点的详细讲解。

在本实例中, 当用户滑动滑块时, 就是实现评分控件中评分的效果。它的实现, 首先需要 slider 滑块控件的值赋值给 rateView 评分控件中的 rate 即评分等级, 代码如下:

```
rateView.rate = slider.value;
```

其次调用 setRate:方法, 实现刷新功能的调用。代码如下:

```
[self refresh];
```

接着实现刷新功能, 在此功能中最重要的是对五角星中值的设置。

```
int size = self.rateViews.count;
for (int i = 0; i < size; i++) {
    View *view = [self.rateViews objectAtIndex:i];
    if (self.rate >= i + 1) {
        view.value = 1;
    } else if (self.rate > i) {
        view.value = self.rate - i;
    } else {
        view.value = 0;
    }
}
```

最后实现五角星的重新绘制。具体的执行流程如图 2.16 所示。

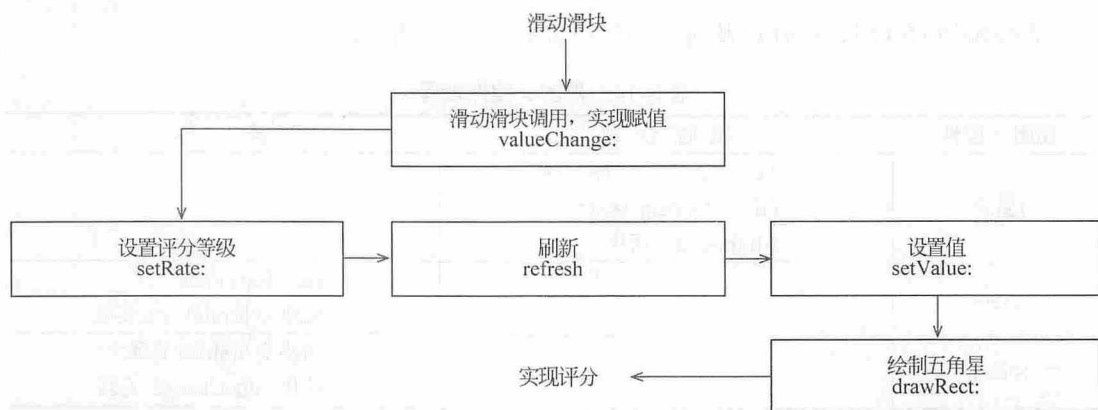


图 2.16 程序执行流程

实例 32 图像的多点点击

【实例描述】

本实例实现的功能是图像的多点点击。当用户点击界面上的红点时，就会出现一个显示有“我是一个小青蛙，快来点我吧！”的标签控件，当再一次点击时，此控件消失。运行效果如图 2.17 所示。

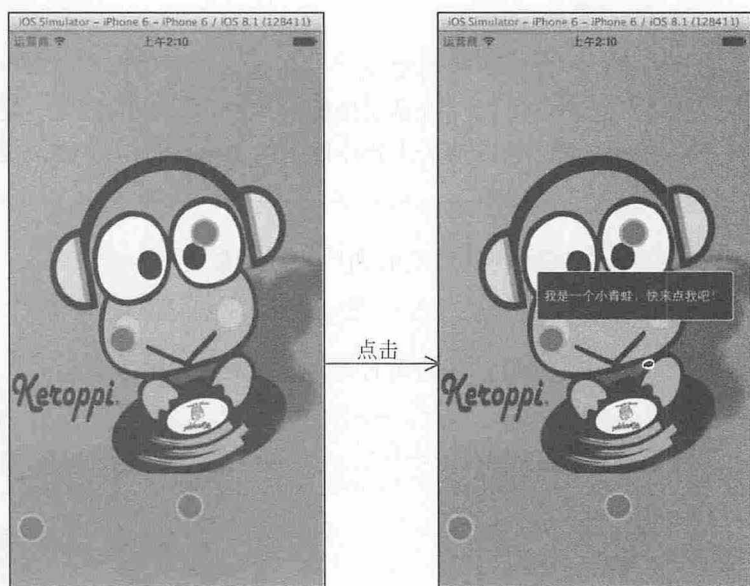


图 2.17 运行效果

【实现过程】

- (1) 创建一个项目，命名为“图像的多点点击”。
- (2) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。

(3) 创建一个基于 UIButton 类的 Dot 类。

(4) 打开 Dot.m 文件, 编写代码, 实现 Dot 的初始化。程序代码如下:

```
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.layer.masksToBounds = YES;
        self.layer.cornerRadius = 15.0;           //设置圆角半径
        self.layer.borderColor = [UIColor colorWithWhite:1.0 alpha:0.5].
        CGColor;                                   //设置边框背景
        self.layer.borderWidth = 3.0;              //设置边框的宽度
    }
    return self;
}
```

(5) 创建一个基于 NSObject 类的 ButtonFactory 类。

(6) 打开 ButtonFactory.h 文件, 编写代码, 实现头文件、属性、方法的声明。程序代码如下:

```
#import <Foundation/Foundation.h>
#import "Dot.h"
@interface ButtonFactory : NSObject
@property(nonatomic, strong)NSMutableDictionary *buttonDic; //属性
-(Dot *)addButtonViewWithKey: (NSInteger)key;                //方法
@end
```

(7) 打开 ButtonFactory.m 文件, 编写代码, 对声明的方法进行实现(即为字典添加 Dot 按钮对象)。程序代码如下:

```
-(Dot *)addButtonViewWithKey: (NSInteger)key
{
    if (_buttonDic == nil){
        self.buttonDic = [NSMutableDictionary dictionary]; //创建可变字典
    }
    Dot *button = [_buttonDic objectForKey:@(key)];
    if (button == nil) {
        button = [Dot buttonWithTypeCustom]; //实例化对象
        button.backgroundColor = [UIColor redColor]; //设置背景颜色
        [_buttonDic setObject:button forKey:@(key)];
    }
    return button;
}
```

(8) 打开 ViewController.h 文件, 编写代码, 实现头文件、数据结构的定义、实例变量以及属性的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
//头文件
#import "Dot.h"
#import "ButtonFactory.h"
//数据结构的定义
typedef struct{
    NSInteger buttonTag;
    __unsafe_unretained UIButton *buttonView;
    CGRect area;
} ButtonState;
```

```

@interface ViewController : UIViewController{
    NSInteger _bTag;
}
//属性
@property(nonatomic,strong)UIImageView *imageView;
@property(nonatomic,strong)UILabel *tipsLabel;
@end

```

(9) 打开 ViewController.m 文件，编写代码，实现图像的多点点击功能。使用的方法如表 2-15 所示。

表 2-15 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
addMoreButtonWithList:	添加按钮
showTips:	显示标签

viewDidLoad 方法在视图加载后调用，实现了对视图的初始化效果。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    //创建并设置图像对象
    _imageView = [[UIImageView alloc] initWithFrame:self.view.bounds];
    _imageView.image = [UIImage imageNamed:@"1.jpg"]; //设置图像视图显示的图像
    _imageView.userInteractionEnabled = YES;
    [self.view addSubview:_imageView]; //添加视图对象
    ButtonFactory *factory = [[ButtonFactory alloc] init];
    NSMutableArray *buttonList = [NSMutableArray array]; //创建可变数组
    //循环添加对象
    for (int i = 0; i < 4; i++) {
        Dot *buttonView = [factory addButtonViewWithKey:i];
        CGRect screenBounds = [[UIScreen mainScreen] bounds]; //获取边界
        //生成随机数为 x 和 y 赋值
        CGFloat x = (arc4random() % (NSInteger)(screenBounds.size.width-30.0));
        CGFloat y = (arc4random() % (NSInteger)(screenBounds.size.height-30.0));
        CGFloat size = 30.0f;
        ButtonState buttonState;
        buttonState.buttonTag = i; //设置 tag 值
        buttonState.buttonView = buttonView;
        buttonState.area = CGRectMake(x, y, size, size);
        //添加对象
        [buttonList addObject:[NSValue value:&buttonState withObjCType:@encode(
            ButtonState)]];
    }
    [self addMoreButtonWithList:buttonList];
    //创建并设置标签对象
    _tipsLabel = [[UILabel alloc] initWithFrame:CGRectMake(60, 0, 200, 50)];
    _tipsLabel.backgroundColor = [UIColor blackColor]; //设置背景颜色
    _tipsLabel.textColor = [UIColor whiteColor];
    _tipsLabel.font = [UIFont systemFontOfSize:13]; //设置字体
    _tipsLabel.lineBreakMode = NSLineBreakByWordWrapping;
    _tipsLabel.numberOfLines = 0; //设置行数
    _tipsLabel.text = @" 我是一个小青蛙，快来点我吧！";
    _tipsLabel.layer.masksToBounds = YES;
}

```

```

    _tipsLabel.layer.borderColor = [UIColor whiteColor].CGColor;
                                                    //设置边框颜色
    _tipsLabel.layer.borderWidth = 1.0;
    _tipsLabel.layer.cornerRadius = 3.0
                                                    //设置圆角半径;
    _tipsLabel.hidden = YES;
    [self.view addSubview:_tipsLabel];
}

```

addMoreButtonWithList:方法实现对按钮的添加,程序代码如下:

```

-(void)addMoreButtonWithList:(NSMutableArray *)list
{
    //遍历,添加按钮
    for (NSValue *stateValue in list)
    {
        ButtonState state;
        [stateValue getValue:&state];
        UIButton *buttonView = state.buttonView;
                                                    //实例化对象
        CGRect area = state.area;
        buttonView.frame = area;
                                                    //设置框架
        buttonView.tag = state.buttonTag;
                                                    //设置tag值
        [buttonView addTarget:self action:@selector(showTips:) forControlEvents:UIControlEventTouchUpInside];
                                                    //添加动作
        [self.view addSubview:buttonView];
    }
}

```

showTips:方法实现单击按钮后显示或者隐藏标签的功能。程序代码如下:

```

-(void)showTips:(UIButton *)button
{
    int tag = button.tag;
    //判断_bTag 是否不等于 tag 值
    if (_bTag != tag) {
        _bTag = tag;
        CGRect frame = _tipsLabel.frame;
                                                    //获取框架
        float x = ((button.frame.origin.x + 200) > 320) ? ((button.frame.origin.x - 170) < 0 ? 60 : (button.frame.origin.x - 170)) : (button.frame.origin.x);
        float y = ((button.frame.origin.y - 55) < 0) ? (button.frame.origin.y + 35) : (button.frame.origin.y - 55);
        frame.origin.x = x;
                                                    //设置x值
        frame.origin.y = y;
        _tipsLabel.hidden = NO;
                                                    //隐藏标签
        _tipsLabel.frame = frame;
                                                    //设置框架
    }else {
        //判断标签对象_tipsLabel 是否隐藏
        if (_tipsLabel.isHidden) {
            _tipsLabel.hidden = NO;
        }else {
            _tipsLabel.hidden = YES;
        }
    }
}
}

```

【代码解析】

本实例关键功能是多点的添加以及点击某一点后,标签位置的改变。下面依次讲解这

两个知识点。

1. 多点的添加

在本实例中多点的添加首先需要将这多个点添加到一个数组中，代码如下：

```
for (int i =0; i<4; i++) {
    Dot *buttonView = [factory addButtonViewWithKey:i];
    CGRect screenBounds = [[UIScreen mainScreen] bounds];
    CGFloat x = (arc4random() % (NSInteger)(screenBounds.size.width-30.0));
    CGFloat y = (arc4random() % (NSInteger)(screenBounds.size.height-30.0));
    CGFloat size = 30.0f;
    ButtonState buttonState;
    buttonState.buttonTag = i;
    buttonState.buttonView = buttonView;
    buttonState.area = CGRectMake(x, y, size, size);
    [buttonList addObject:[NSValue value:&buttonState withObjCType:@encode
        (ButtonState)]];
}
```

其次需要将数据中的对象逐个遍历，并添加到当前的视图中，代码如下：

```
for (NSValue *stateValue in list) {
    ButtonState state;
    .....
    [buttonView addTarget:self action:@selector(showTips:) forControlEvents:
        UIControlEventTouchUpInside];
    [self.view addSubview:buttonView];
}
```

2. 点击某一点后，标签位置的改变

在本实例中，标签位置的改变是通过改变 frame 实现的，代码如下：

```
CGRect frame = _tipsLabel.frame;
float x = ((button.frame.origin.x + 200) > 320 ) ? ((button.frame.origin.x
- 170) < 0 ? 60 : (button.frame.origin.x - 170)) : (button.frame.origin.x);
float y = ((button.frame.origin.y -55) < 0) ? (button.frame.origin.y + 35) :
(button.frame.origin.y -55);
frame.origin.x = x;
frame.origin.y = y;
_tipsLabel.hidden = NO;
_tipsLabel.frame = frame;
```

实例 33 裁剪图像

【实例描述】

在图像编辑时，常常会出现裁剪图像的功能，那么这些功能是如何实现的呢。本实例就为读者实现一个裁剪图像的功能。在界面中的一个蓝色的框就是裁剪区域，当用户对图像进行放大、旋转或者移动后，单击 Crop 按钮，就会将裁剪区域以外的图像进行裁剪，并进行显示。效果如图 2.18 所示。

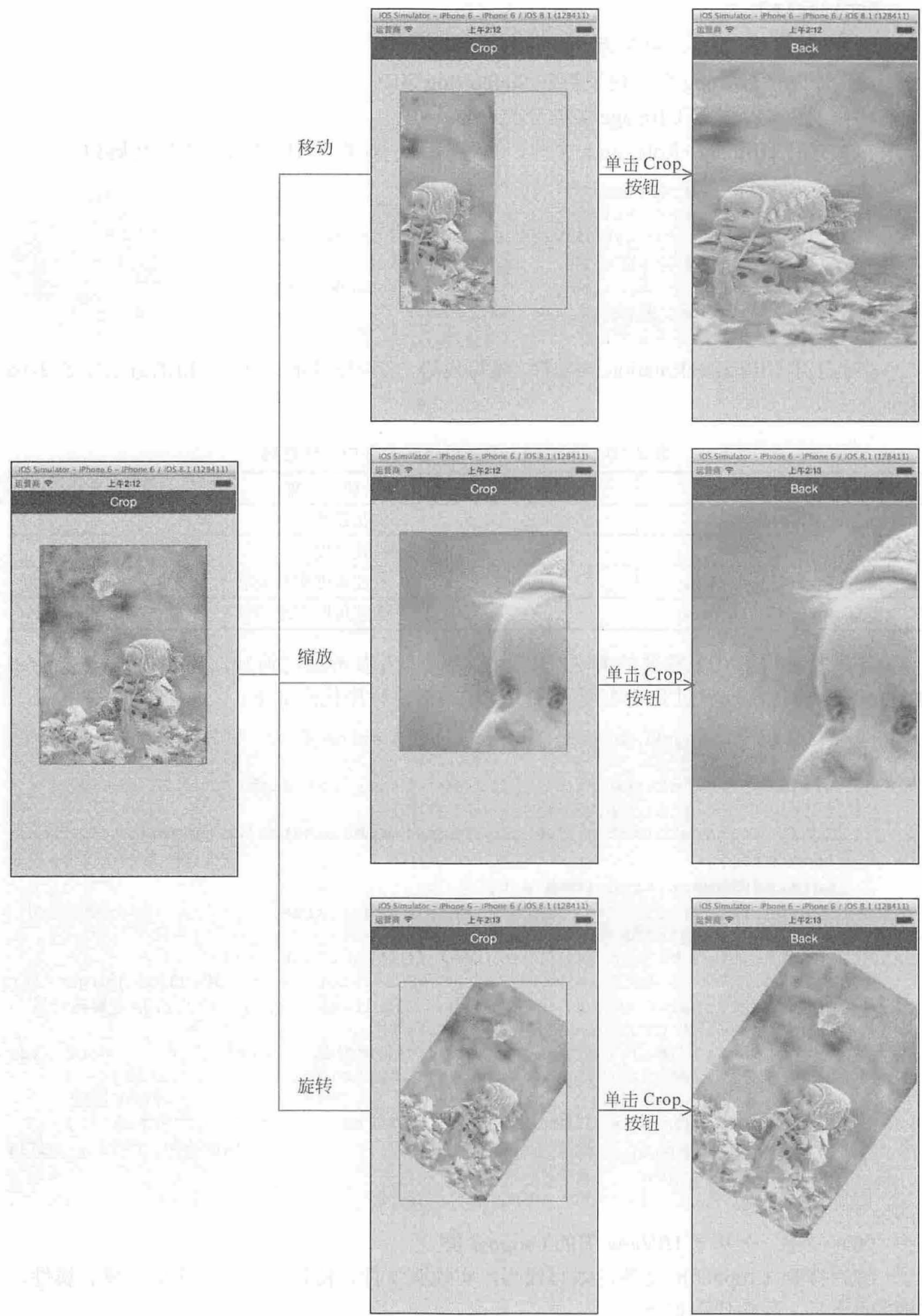


图 2.18 运行效果

【实现过程】

- (1) 创建一个项目，命名为“裁剪图像”。
- (2) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIImage 类的分类 Rotation。
- (4) 打开 UIImage+Rotation.h 文件，编写代码，实现方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface UIImage (Rotation)
- (UIImage *)imageRotatedByRadians:(CGFloat) radians;
    //通过弧度实现旋转
- (UIImage *)imageRotatedByDegrees:(CGFloat) degrees;
    //通过角度实现旋转
@end
```

(5) 打开 UIImage+Rotation.m 文件，编写代码，实现图像的旋转。使用的方法如表 2-16 所示。

表 2-16 UIImage+Rotation.m文件中方法总结

方 法	功 能
DegreesToRadians	获取弧度
RadiansToDegrees	获取角度
imageRotatedByRadians:	通过弧度实现旋转
imageRotatedByDegrees:	通过角度实现旋转

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，imageRotatedByDegrees:方法实现通过角度实现图像的旋转功能。程序代码如下：

```
- (UIImage *)imageRotatedByDegrees:(CGFloat) degrees
{
    UIView *rotatedViewBox = [[UIView alloc] initWithFrame:CGRectMakeMake
    (0,0,self.size.width, self.size.height)];
    CGAffineTransform t = CGAffineTransformMakeRotation(DegreesToRadians
    (degrees)); //旋转
    rotatedViewBox.transform = t; //改变
    CGSize rotatedSize = rotatedViewBox.frame.size; //获取尺寸
    UIGraphicsBeginImageContext(rotatedSize);
    CGContextRef bitmap = UIGraphicsGetCurrentContext();
    CGContextTranslateCTM(bitmap, rotatedSize.width/2, rotatedSize.height/2);
    CGContextRotateCTM(bitmap, DegreesToRadians(degrees)); //指定旋转角度
    CGContextScaleCTM(bitmap, 1.0, -1.0);
    CGContextDrawImage(bitmap, CGRectMake(-self.size.width / 2, -self.size.
    height / 2, self.size.width, self.size.height), [self CGImage]); //绘制图像
    UIImage *resImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext(); //结束绘图，并关闭绘图环境
    return resImage;
}
```

- (6) 创建一个基于 UIView 类的 Cropper 类。
- (7) 打开 Cropper.h 文件，编写代码，实现头文件、协议、对象、实例变量、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
//头文件
```

```

#import "UIImage+Rotation.h"
@class Cropper;
//协议
@protocol CropperDelegate <NSObject>
- (void)imageCropper:(Cropper *)cropper didFinishCroppingWithImage:(UIImage *)
image;
@end
@interface Cropper : UIView{
    //对象
    UIImageView *imageView;
    CGSize _originalImageViewSize;
}
//属性
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, retain) UIImage *croppedImage;
@property (nonatomic, retain) UIImageView *imageView;
@property (nonatomic, assign) id <CropperDelegate> delegate;
//方法
- (void)setup; //创建
- (void)finishCropping;
- (void)reset; //重置
@end

```

(8) 打开 Cropper.m 文件, 编写代码, 实现裁剪功能, 使用的方法如表 2-17 所示。

表 2-17 Cropper.m文件中方法总结

方 法	功 能
setup	创建
initWithFrame:	初始化
moveImage:	移动图像
scaleImage:	缩放图像
rotateImage:	旋转图像
setImage:	设置图像
finishCropping	结束裁剪
reset	重置

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, setup 方法实现对一些手势识别器对象的创建。程序代码如下:

```

- (void)setup
{
    self.clipsToBounds = YES;
    self.backgroundColor = [UIColor clearColor]; //设置背景颜色
    self.imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0.0, 0.0,
self.frame.size.width, self.frame.size.height)];
    imageView.userInteractionEnabled = YES;
    [self addSubview:imageView]; //添加视图对象
    //创建并设置旋转手势识别器对象
    UIRotationGestureRecognizer *rotateGes = [[UIRotationGestureRecognizer
alloc] initWithTarget:self action:@selector(rotateImage:)];
    //创建旋转手势识别器对象
    [imageView addGestureRecognizer:rotateGes];
    //创建并设置缩放手势识别器对象
    UIPinchGestureRecognizer *scaleGes = [[UIPinchGestureRecognizer alloc]
initWithTarget:self action:@selector(scaleImage:)];
}

```

```

//创建缩放手势识别器对象
[imageView addGestureRecognizer:scaleGes];
//创建并设置移动手势识别器对象
UIPanGestureRecognizer *moveGes = [[UIPanGestureRecognizer alloc]
initWithTarget:self action:@selector(moveImage:)]; //创建移动手势识别器对象
[moveGes setMinimumNumberOfTouches:1];
[moveGes setMaximumNumberOfTouches:1];
[imageView addGestureRecognizer:moveGes];
}

```

moveImage:、scaleImage:、rotateImage:方法实现对图像进行的移动、缩放以及旋转操作。程序代码如下:

```

//移动
- (void)moveImage:(UIPanGestureRecognizer *)sender
{
CGPoint translatedPoint = [sender translationInView:self];
//判断 state 是否为 UIGestureRecognizerStateBegan
if([sender state] == UIGestureRecognizerStateBegan) {
    _lastTransX = 0.0;
    _lastTransY = 0.0;
}
CGAffineTransform trans = CGAffineTransformMakeTranslation(translatedPoint.x - _lastTransX, translatedPoint.y - _lastTransY); //平移
CGAffineTransform newTransform = CGAffineTransformConcat(imageView.transform, trans);
_lastTransX = translatedPoint.x; //获取 x 值
_lastTransY = translatedPoint.y; //获取 y 值
imageView.transform = newTransform;
}

//缩放
- (void)scaleImage:(UIPinchGestureRecognizer *)sender
{
    //判断 state 是否为 UIGestureRecognizerStateBegan
    if([sender state] == UIGestureRecognizerStateBegan) {
        _lastScale = 1.0;
        return;
    }
    CGFloat scale = [sender scale]/_lastScale;
    CGAffineTransform currentTransform = imageView.transform;
    CGAffineTransform newTransform = CGAffineTransformScale(currentTransform, scale, scale); //缩放
    [imageView setTransform:newTransform];
    _lastScale = [sender scale];
}

//旋转
- (void)rotateImage:(UIRotationGestureRecognizer *)sender
{
    //判断 state 是否为 UIGestureRecognizerStateBegan
    if([sender state] == UIGestureRecognizerStateEnded) {
        _lastRotation = 0.0;
        return;
    }
    CGFloat rotation = -_lastRotation + [sender rotation];
    CGAffineTransform currentTransform = imageView.transform;
    CGAffineTransform newTransform = CGAffineTransformRotate(currentTransform, rotation); //旋转
    [imageView setTransform:newTransform];
}

```



```

    _lastRotation = [sender rotation];
}

```

setImage:方法实现对图像的设置,重点是对位置的设置。程序代码如下:

```

- (void)setImage:(UIImage *)image
{
    if (_image != image) {
        _image = image;
    }
    float _imageScale = self.frame.size.width / image.size.width;
    //设置框架
    self.imageView.frame = CGRectMake(0, 0, image.size.width*_imageScale,
    image.size.height*_imageScale);
    _originalImageViewSize = CGSizeMake(image.size.width*_imageScale,
    image.size.height*_imageScale);
    imageView.image = image; //设置图像
    imageView.center = CGPointMake(self.frame.size.width/2.0, self.frame.
size.height/2.0); //设置中心位置
}

```

finishCropping 方法实现结束裁剪功能。程序代码如下:

```

- (void)finishCropping {
    //获取 x 轴上的缩放值
    float zoomScale = [[self.imageView.layer valueForKeyPath:@"transform.
    scale. x"] floatValue]; //获取 z 轴上的旋转值
    float rotate = [[self.imageView.layer valueForKeyPath:@"transform.
    rotation.z"] floatValue];
    float _imageScale = _image.size.width/_originalImageViewSize.width;
    //实例化对象
    CGSize cropSize = CGSizeMake(self.frame.size.width/zoomScale, self.frame.
    size.height/zoomScale);
    CGPoint cropperViewOrigin = CGPointMake((0.0 - self.imageView.frame.
    origin.x)/zoomScale, (0.0 - self.imageView.frame.origin.y)/zoomScale);
    if((NSInteger)cropSize.width % 2 == 1){
        cropSize.width = ceil(cropSize.width); //设置宽度
    }
    if((NSInteger)cropSize.height % 2 == 1){
        cropSize.height = ceil(cropSize.height); //设置高度
    }
    //创建裁剪的矩形区域
    CGRect CropRectInImage = CGRectMake( (NSInteger) (cropperViewOrigin.x*_
imageScale), (NSInteger) ( cropperViewOrigin.y*_imageScale), (NSInteger)
(cropSize.width*_imageScale), (NSInteger) (cropSize.height*_imageScale));
    UIImage *rotInputImage = [self.image imageRotatedByRadians:rotate];
    //实例化对象
    //实现截取
    CGImageRef tmp = CGImageCreateWithImageInRect([rotInputImage CGImage],
CropRectInImage);
    self.croppedImage = [UIImage imageWithCGImage:tmp scale:self.image.scale
    orientation:self.image.imageOrientation];
    CGImageRelease(tmp); //释放位图上下文
}

```

(9) 打开 ViewController.h 文件,编写代码,实现头文件、插座变量以及动作的声明。程序代码如下:

```

#import <UIKit/UIKit.h>

```

```
#import "Cropper.h"
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet Cropper *cropper;
    IBOutlet UIImageView *result;
    IBOutlet UIButton *btn;
}
- (IBAction)crop:(id)sender;
@end
```

(10) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 2.19 所示。

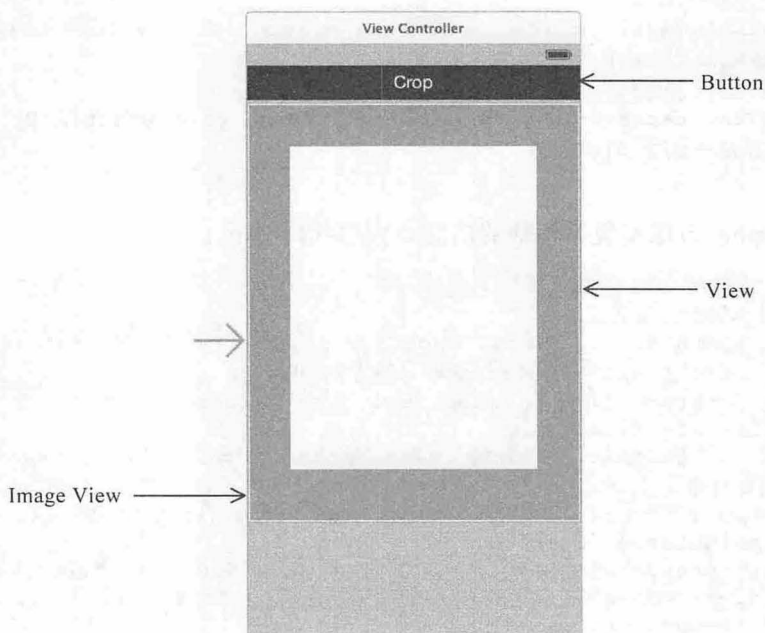


图 2.19 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-18 所示。

表 2-18 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 灰色	
Button	Title: Crop Font: System 17.0 Text Color: 白色 Background: 黑色	与插座变量btn关联 与动作crop:关联
View		Class: Cropper 与插座变量cropper关联
Image View		与插座变量result关联

(11) 打开 ViewController.m 文件，编写代码，实现对界面的设计以及单击按钮后实现的裁剪功能。程序代码如下：

```
- (void)viewDidLoad
```

```

{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    cropper.layer.borderWidth = 1.0;           //设置边框的宽度
    cropper.layer.borderColor = [UIColor blueColor].CGColor; //设置边框的颜色

    [cropper setup];
    cropper.image = [UIImage imageNamed:@"1.jpg"]; //设置图像
}

- (IBAction)crop:(id)sender {
    if ([btn.currentTitle isEqualToString:@"Crop"]) {
        [cropper finishCropping];
        result.image = cropper.croppedImage; //设置图像
        cropper.hidden = YES;                //隐藏 cropper 对象
        [btn setTitle:@"Back" forState:UIControlStateNormal]; //设置标题
        [btn setTitle:@"Back" forState:UIControlStateHighlighted]; //设置标签
    } else {
        [cropper reset];
        cropper.hidden = NO;                  //显示 cropper 对象
        [btn setTitle:@"Crop" forState:UIControlStateNormal];
        [btn setTitle:@"Crop" forState:UIControlStateHighlighted];
        result.image = nil;                  //设置图像
    }
}
}

```

【代码解析】

本实例关键功能是图像的裁剪功能。下面就是这个知识点的详细讲解。

实现裁剪图像的方法有很多种,其中之一是使用 `CGImage` 的 `CGImageCreateWithImageInRect` 函数实现,其语法形式如下:

```

CGImageRef CGImageCreateWithImageInRect (
    CGImageRef image,
    CGRect rect
);

```

其中, `CGImageRef image` 表示提取的子图像; `CGRect rect` 表示一个矩形区域。在此代码中就是使用了 `CGImageCreateWithImageInRect` 函数实现的图像裁剪功能,代码如下:

```

CGImageRef tmp = CGImageCreateWithImageInRect([rotInputImage CGImage],
    CropRectinImage);

```

其中, `[rotInputImage CGImage]` 表示提取的子图像, `CropRectinImage` 表示一个矩形区域。

实例 34 图像主要颜色的提取

【实例描述】

本实例实现的功能是提取图像中主要的颜色,并作为背景。当选择分段控件中的某段后,就会出现相应的图像以及背景效果。运行效果如图 2.20 所示。

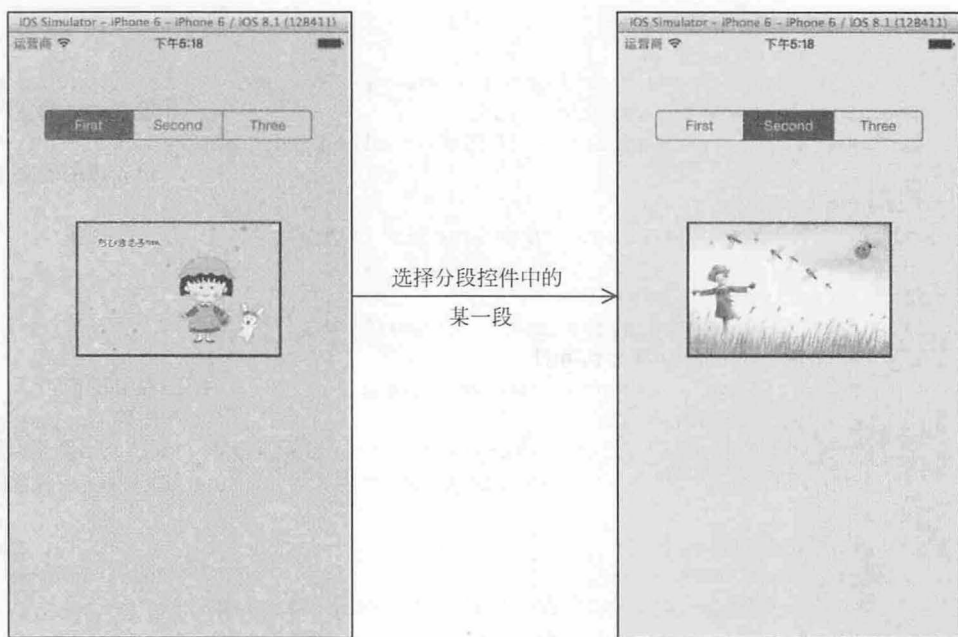


图 2.20 运行效果

【实现过程】

- (1) 创建一个项目，命名为“图像主要颜色的提取”。
- (2) 添加图像 1.jpg、2.jpg、3.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIImage 类的分类 MainColor。
- (4) 打开 UIImage+MainColor.h 文件，编写代码，实现方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface UIImage (MainColor)
-(UIColor*)mostColor;
@end
```

- (5) 打开 UIImage+MainColor.m 文件，编写代码，实现主要颜色的获取。程序代码如下：

```
-(UIColor*)mostColor{
    //把图片缩小可加快设备的计算速度，但越小，误差可能越大
    CGSize thumbSize=CGSizeMake(50, 50);
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();//创建色彩空间
    CGContextRef context = CGBitmapContextCreate(NULL,thumbSize.width,
    thumbSize.height,8,thumbSize.width*4,colorSpace,kCGImageAlphaPremul
    tipliedLast | kCGBitmapByteOrder32Big); //创建位图上下文
    CGRect drawRect = CGRectMake(0, 0, thumbSize.width, thumbSize.height);
    CGContextDrawImage(context, drawRect, self.CGImage); //绘制图像
    CGColorSpaceRelease(colorSpace); //释放色彩空间
    //获取每个点的像素值
    unsigned char* data = CGBitmapContextGetData (context);
    //判断 data 是否为空
    if (data == NULL)
        return nil;
```



```

NSMutableDictionary *cls=[NSMutableDictionary dictionaryWithCapacity:thumbSize.width*thumbSize.height];
//循环添加对象
for (int x=0; x<thumbSize.width; x++) {
    //循环
    for (int y=0; y<thumbSize.height; y++) {
        //为变量赋值
        int offset = 4*(x*y);
        int red = data[offset];
        int green = data[offset+1];
        int blue = data[offset+2];
        int alpha = data[offset+3];
        NSArray *clr=@(red),@(green),@(blue),@(alpha)]; //创建数组
        [cls addObject:clr]; //添加对象
    }
}
CGContextRelease(context);
//找到出现次数最多的那个颜色
NSEnumerator *enumerator = [cls objectEnumerator];
NSArray *curColor = nil; //实例化数组
NSArray *MaxColor=nil;
NSUInteger MaxCount=0;
//循环
while ( (curColor = [enumerator nextObject]) != nil )
{
    NSUInteger tmpCount = [cls countForObject:curColor];
    //判断 tmpCount 是否小于 MaxCount
    if ( tmpCount < MaxCount )
        continue;
    MaxCount=tmpCount;
    MaxColor=curColor;
}
return [UIColor colorWithRed:([MaxColor[0] intValue]/255.0f) green:([MaxColor[1] intValue]/255.0f) blue:([MaxColor[2] intValue]/255.0f) alpha:([MaxColor[3] intValue]/255.0f)]; //返回颜色
}

```

(6) 打开 ViewController.h 文件, 编写代码, 实现头文件、插座变量以及动作的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "UIImage+MainColor.h"
@interface ViewController : UIViewController
    //插座变量
    IBOutlet UISegmentedControl *segmentedControl;
    IBOutlet UIImageView *imageView;
}
- (IBAction)Select:(id)sender;
@end

```

(7) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 2.21 所示。

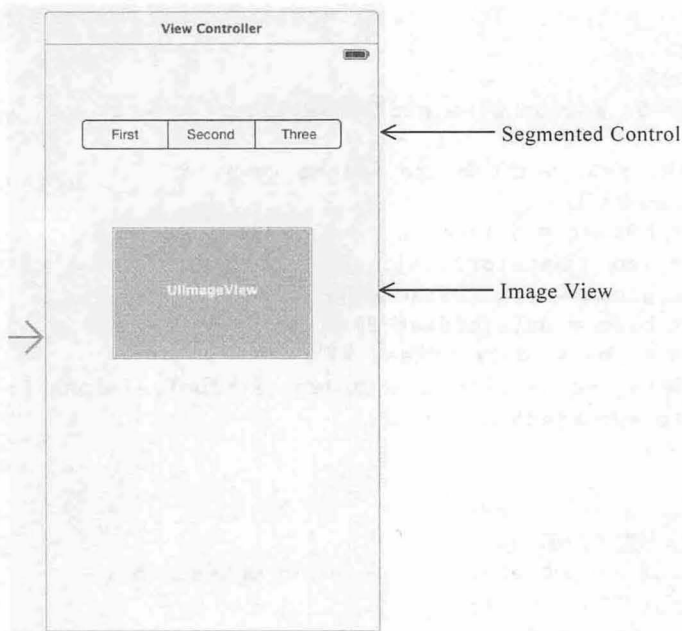


图 2.21 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-19 所示。

表 2-19 视图、控件设置

视图、控件	属 性 设 置	其 他
Segmented Contol	Segments: 3	与插座变量segmentedControl关联 与动作Select关联
	Segment: Segment 0 Title: First	
	Segment: Segment 1 Title: Second	
	Segment: Segment 2 Title: Three	
	Tint: 棕色	
Image View		与插座变量imageView关联

(8) 打开 ViewController.m 文件，编写代码，实现设计界面的初始化，选择分段控件中的某一段后提取相应图像的主要颜色要作为背景色。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    UIImageView.layer.borderWidth=2;
    UIImage *img=[UIImage imageNamed:@"1.jpg"];
    imageView.image=img;                                //设置图像
    UIColor *most=[img mostColor];
    self.view.backgroundColor=most;                      //设置背景颜色
}
//选择分段控件中的某一段
- (IBAction)Select:(id)sender {
    NSInteger select=segmentedControl.selectedSegmentIndex;
                                                                //获取当前段的索引值
    //判断 select 是否为 0
    if (select==0) {
```

```

UIImage *img=[UIImage imageNamed:@"1.jpg"];
imageView.image=img;                                //设置图像
UIColor *most=[img mostColor];
self.view.backgroundColor=most;
}else if (select==1){                                //判断 select 是否为 0
    UIImage *img=[UIImage imageNamed:@"2.jpg"];
    imageView.image=img;                                //设置图像
    UIColor *most=[img mostColor];
    self.view.backgroundColor=most;
}else{
    UIImage *img=[UIImage imageNamed:@"3.jpg"];
    imageView.image=img;                                //设置图像
    UIColor *most=[img mostColor];
    self.view.backgroundColor=most;
}
}

```

【代码解析】

本实例关键功能是像素点的颜色值的获取以及颜色的比较。下面依次讲解这两个知识点。

1. 像素点颜色值的获取

本实例中要获取图像中每一个像素的颜色值，需要使用双重 for 循环实现，代码如下：

```

for (int x=0; x<thumbSize.width; x++) {
    for (int y=0; y<thumbSize.height; y++) {
        int offset = 4*(x*y);
        int red = data[offset];
        int green = data[offset+1];
        int blue = data[offset+2];
        int alpha = data[offset+3];
        NSArray *clr=@[(red),@(green),@(blue),@(alpha)];
        [cls addObject:clr];
    }
}

```

2. 颜色的比较

要实现颜色的比较，首先需要声明两个数组，用来保存当前的颜色以及最大的颜色。还需要定义一个整型变量，用来保存最大计数值。代码如下：

```

NSArray *curColor = nil;
NSArray *MaxColor=nil;
NSUInteger MaxCount=0;

```

最后实现比较功能，代码如下：

```

while ( (curColor = [enumerator nextObject]) != nil )
{
    NSUInteger tmpCount = [cls countForObject:curColor];
    if ( tmpCount < MaxCount )
        continue;
    MaxCount=tmpCount;
    书 MaxColor=curColor;
}

```

实例 35 动物连连看

【实例描述】

连连看受不少读者的喜爱，甚至不少人成为了连连迷。那么它是如何实现的呢？本实例就满足这些连连迷的需要，来实现一个连连看的功能。运行效果如图 2.22 所示。

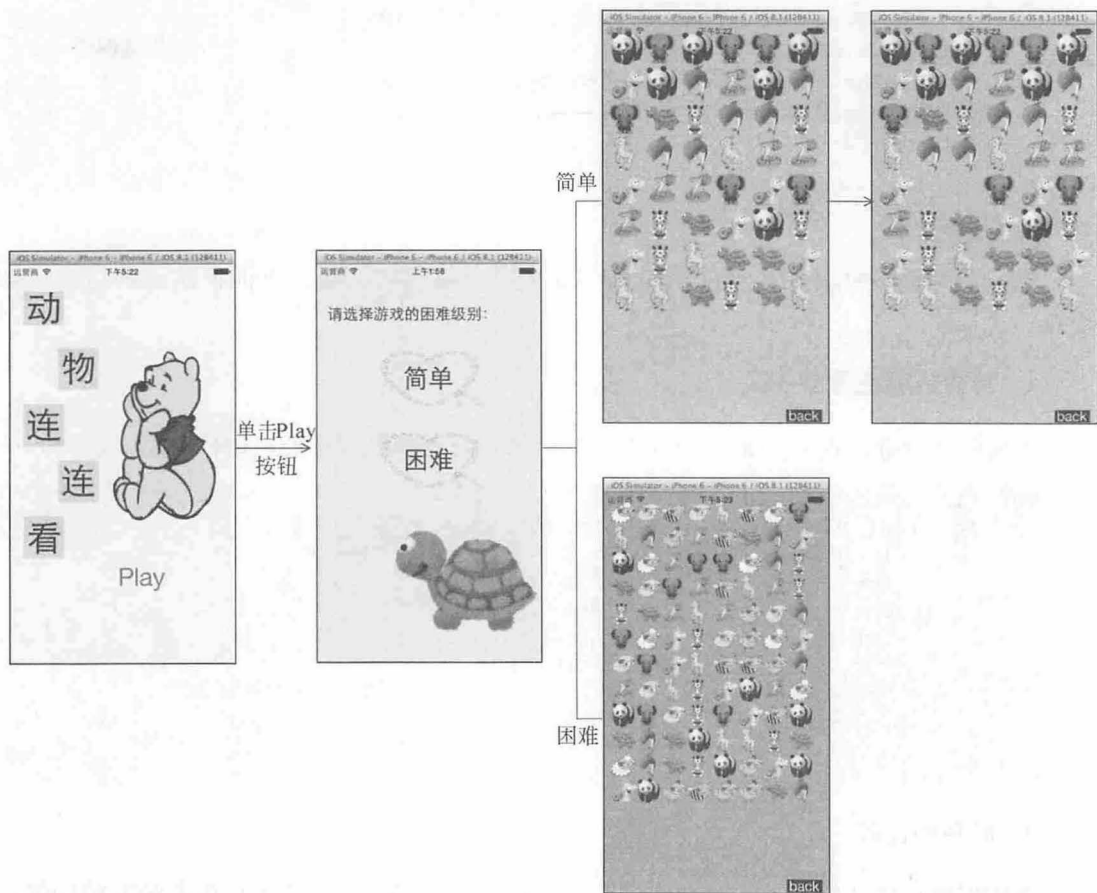


图 2.22 运行效果

【实现过程】

- (1) 创建一个项目，命名为“动物连连看”。
 - (2) 添加图像 bg0.png~bg11.png、bg12.jpg、bg13.png 到创建项目的 Supporting Files 文件夹中。
 - (3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 2.23 所示。
- 需要添加的视图、控件以及对它们的设置如表 2-20 所示。

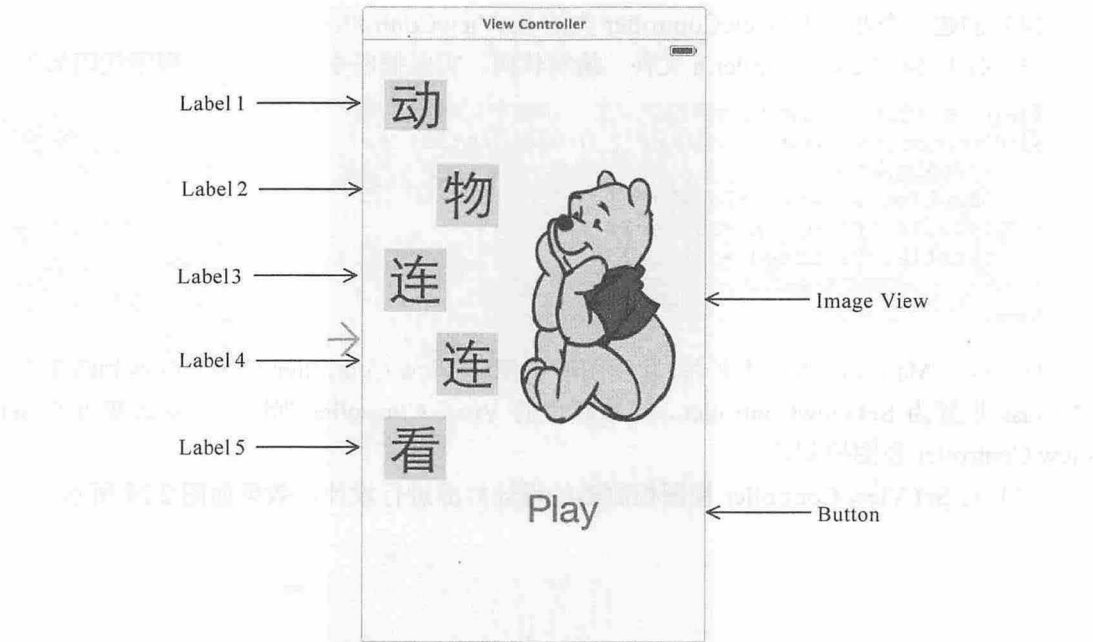


图 2.23 View Controller 视图控制器的设计界面

表 2-20 视图、控件设置

视图、控件	属 性 设 置	其 他
Label1	Text: 动 Font: System 50.0 Alignment: 居中 Background: 浅蓝色	
Label2	Text: 物 Font: System 50.0 Alignment: 居中 Background: 浅蓝色	
Label3	Text: 连 Font: System 50.0 Alignment: 居中 Background: 浅蓝色	
Label4	Text: 连 Font: System 50.0 Alignment: 居中 Background: 浅蓝色	
Label5	Text: 看 Font: System 50.0 Alignment: 居中 Background: 浅蓝色	
Image View	Image: bg12.png	
Button	Title: Play Font: System 36.0	

- (4) 创建一个基于 UIViewController 类的 SetViewController 类。
- (5) 打开 SetViewController.h 文件，编写代码，实现插座变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface SetViewController : UIViewController{
    //声明插座变量
    IBOutlet UILabel *label;
    IBOutlet UIButton *button1;
    IBOutlet UIButton *button2;
}
@end
```

- (6) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 SetViewController。这时新增的 View Controller 视图控制器就变为了 Set View Controller 视图控制器。
- (7) 对 Set View Controller 视图控制器的设计界面进行设计，效果如图 2.24 所示。

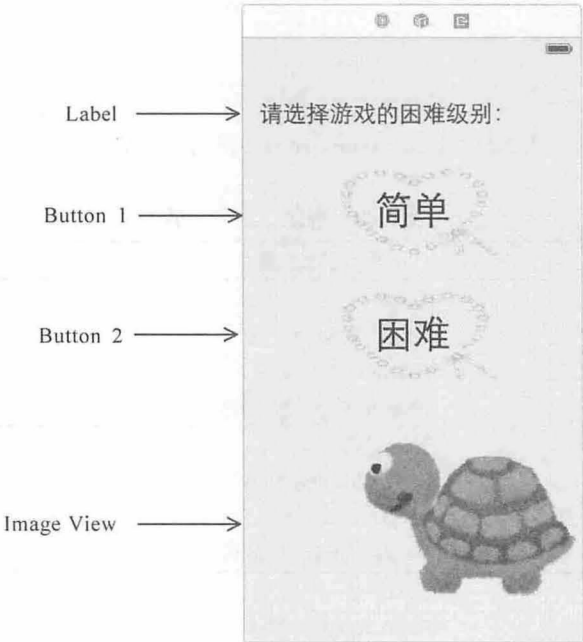


图 2.24 Set View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-21 所示。

表 2-21 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅橘红色	
Label	Text: 请选择游戏的困难级别: Font: System 22.0	与插座变量label关联
Button1	Title: 简单 Font: System 36.0 Text Color: 黑色 Background: bg13.png	与插座变量button1关联

续表

视图、控件	属 性 设 置	其 他
Button2	Title: 困难 Font: System 36.0 Text Color: 黑色 Background: bg13.png	与插座变量button2关联
Image View	Image: bg0.png	

(8) 打开 SetViewController.m 文件, 编写代码, 实现按钮的动画效果。程序代码如下:

```

- (void)viewDidLoad
{
    //隐藏按钮对象
    [button1 setHidden:YES];
    [button2 setHidden:YES];
    [self performSelector:@selector(aa) withObject:self afterDelay:0.2];
    [super viewDidLoad];
    // Do any additional setup after loading the view.
}
//动画效果
- (void)aa{
    CATransition *transition=[CATransition animation];
    transition.type=kCATransitionMoveIn; //设置动画类型
    transition.duration=2.0; //设置动画持续时间
    [button1 setHidden:NO];
    [button1.layer addAnimation:transition forKey:@""]; //添加动作
    CATransition *anima=[CATransition animation];
    anima.type=kCATransitionMoveIn;
    anima.subtype=kCATransitionFromRight; //设置方向
    anima.duration=2.0;
    [button2 setHidden:NO];
    [button2.layer addAnimation:anima forKey:@""]; //添加动画
}

```

(9) 创建一个基于 UIViewController 类的 SimpleViewController 类。

(10) 打开 Main.storyboard 文件, 从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 SimpleViewController。这时新增的 View Controller 视图控制器就变为了 Simple View Controller 视图控制器。

(11) 创建一个基于 UIViewController 类的 TroubleViewController 类。

(12) 打开 Main.storyboard 文件, 从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 TroubleViewController。这时新增的 View Controller 视图控制器就变为了 Trouble View Controller 视图控制器。

(13) 将按钮分别和其他视图控制器的界面进行关联, 如表 2-22 所示。

表 2-22 按钮分别和其他视图控制器的界面进行关联

按 钮	关联的视图控制器界面
Play按钮	与Set View Controller视图控制器的设计界面关联
“简单”按钮	与Simple View Controller视图控制器的设计界面关联
“困难”按钮	与Trouble View Controller视图控制器的设计界面关联

(14) 打开 SimpleViewController.h 文件, 编写代码, 实现头文件、宏定义、数组、对

象、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "SetViewController.h"
#define Row 8      //行数
#define Colum 6    //列数
#define Width 50   //每个 button 的边长
@interface SimpleViewController : UIViewController{
    int map[Row][Colum];
    SetViewController *set;
}
@property (nonatomic ,retain) UIButton *lastButton;//上一次点击的 button;
@property (nonatomic ,retain) UIButton *currentButton; //当前点击的 button;
-(BOOL)canPassFromLastButton:(UIButton *)lastButton ToCurrentButton:
(UIButton *)currentButton;
@end
```

（15）打开 SimpleViewController.m 文件，编写代码，实现连连看的功能。使用的方法如表 2-23 所示。

表 2-23 SimpleViewController.m文件中方法总结

方 法	功 能
getXFromButtonTag:	获取x
getYFromButtonTag:	获取y
checkPathWithX1: andX2: throughY:	检测路径
checkPathWithY1: andY2: throughX:	检测路径
canPassFromLastButton: ToCurrentButton:	限定所有的判断
buttonClicked:	触发当点击到动物时的方法
loadView	加载视图
anniu:	切换视图

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，canPassFromLast Button: ToCurrentButton:方法实现限定所有的判断。由于此方法过程很多，下面将分步进行讲解。首先根据 tag 值获取两点击的坐标。程序代码如下：

```
//第一个点的横纵坐标
int x1 = [self getXFromButtonTag:lastButton.tag];
int y1 = [self getYFromButtonTag:lastButton.tag];
//第二个点的横纵坐标。
int x2 = [self getXFromButtonTag:currentButton.tag];
int y2 = [self getYFromButtonTag:currentButton.tag];
```

其次判断点击的两个点是不是同一个点。程序代码如下：

```
if (x1==x2&& y1==y2) {
    return NO;
}
```

然后判断这两个点是不是同一行，代码如下：

```
else if (y1 == y2) {
    //第一行或是最后一行的
    if ((y1 == 0) || (y1 == Row-1)) {
        return YES;
    }
}
```



```

        return YES;
    }
} else if (y2==Row-1){
    //判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:y2+1 throughX:x1]) {
        return YES;
    }
}
//下面进行两个拐点的判断的写法。
//横向的比较。
for (int i=0; i<Colum; i++) {
    if (i<x1) {
        //判断有没有别的按钮
        if ([self checkPathWithX1:-1 andX2:x1 throughY:y1]&&[self
            checkPathWithX1:-1 andX2:x2 throughY:y2]) {
            return YES;
        }
        //判断有没有别的按钮
        if ([self checkPathWithX1:i-1 andX2:x1 throughY:y1]
            &&[self checkPathWithX1:i-1 andX2:x2 throughY:y2]&&[self
            checkPathWithY1:y1-1 andY2:y2+1 throughX:i]) {
            return YES;
        }
    } else if (i>x1&&i<x2){
        //判断有没有别的按钮
        if ([self checkPathWithX1:x1 andX2:i+1 throughY:y1]
            &&[self checkPathWithX1:i-1 andX2:x2 throughY:y2]&&[self
            checkPathWithY1:y1-1 andY2:y2+1 throughX:i]) {
            return YES;
        }
    } else if (i>x2){
        //判断有没有别的按钮
        if ([self checkPathWithX1:x1 andX2:Colum throughY:y1]
            &&[self checkPathWithX1:x2 andX2:Colum throughY:y2]) {
            return YES;
        }
        //判断有没有别的按钮
        if ([self checkPathWithX1:x1 andX2:i+1 throughY:y1]
            &&[self checkPathWithX1:x2 andX2:i+1 throughY:y2]&&[self
            checkPathWithY1:y1-1 andY2:y2+1 throughX:i]) {
            return YES;
        }
    }
}
//下面进行纵向的比较
for (int i = 0; i<Row; i++) {
    if (i<y1) {
        //判断有没有别的按钮
        if ([self checkPathWithY1:-1 andY2:y1 throughX:x1]&&[self
            checkPathWithY1:-1 andY2:y2 throughX:x2]) {
            return YES;
        }
        //判断有没有别的按钮
        if ([self checkPathWithY1:i-1 andY2:y1 throughX:x1]
            &&[self checkPathWithY1:i-1 andY2:y2 throughX:x2]&&[self
            checkPathWithX1:x1-1 andX2:x2+1 throughY:i]) {
            return YES;
        }
    } else if (i>y1&&i<y2){

```



```
//判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:i+1 throughX:x1]
        &&[self checkPathWithY1:i-1 andY2:y2 throughX:x2]&&[self
            checkPathWithX1:x1-1 andX2:x2+1 throughY:i]) {
        return YES;
    }
} else if(i>y2){
//判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:Row throughX:x1]
        &&[self checkPathWithY1:y2 andY2:Row throughX:x2]) {
        return YES;
    }
//判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:i+1 throughX:x1]&&
        [self checkPathWithY1:y2 andY2:i+1 throughX:x2]&&[self
            checkPathWithX1:x1-1 andX2:x2+1 throughY:i]) {
        return YES;
    }
}
}
```

如果第二个点在第一个点的右下角，那么它的比较代码如下：

```

if (x1>x2) {
    //判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:y2+1 throughX:x1]&&[self
checkPathWithX1:x2 andX2:x1+1 throughY:y2])||([self checkPathWithY1:y1-1
andY2:y2 throughX:x2]&&[self checkPathWithX1:x2-1 andX2:x1 throughY:y1]))
{
        return YES;
    }
    if (x1==Colum-1&&y1==0) {
        //判断有没有别的按钮
        if ([self checkPathWithX1:x2 andX2:x1+1 throughY:y2]) {
            return YES;
        }
        //判断有没有别的按钮
        if ([self checkPathWithY1:y1-1 andY2:y2 throughX:x2]) {
            return YES;
        }
    }else if (x1==Colum-1){
        //判断有没有别的按钮
        if ([self checkPathWithX1:x2 andX2:x1+1 throughY:y2]) {
            return YES;
        }
    }else if (y1==0){
        //判断有没有别的按钮
        if ([self checkPathWithY1:y1-1 andY2:y2 throughX:x2]) {
            return YES;
        }
    }
}
if (x2==0&&y2==Row-1) {
    //判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:y2+1 throughX:x1]) {
        return YES;
    }
    //判断有没有别的按钮
    if ([self checkPathWithX1:x2-1 andX2:x1 throughY:y1]) {
        return YES;
    }
}

```

```

    }
} else if (x2 == 0) {
    //判断有没有别的按钮
    if ([self checkPathWithX1:x2-1 andX2:x1 throughY:y1]) {
        return YES;
    }
} else if (y2 == Row-1) {
    //判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:y2+1 throughX:x1]) {
        return YES;
    }
}

//判断两个拐点的情形
for (int i=0; i<Column; i++) {
    if (i<x2) {
        //判断有没有别的按钮
        if ([self checkPathWithX1:-1 andX2:x1 throughY:y1]&&[self
checkPathWithX1:-1 andX2:x2 throughY:y2]) {
            return YES;
        }
        //判断有没有别的按钮
        if ([self checkPathWithX1:i-1 andX2:x1 throughY:y1]&&[self
checkPathWithX1:i-1 andX2:x2 throughY:y2]&&[self checkPath
WithY1:y1-1 andY2:y2+1 throughX:i]) {
            return YES;
        }
    } else if (i>x2&&i<x1){
        //判断有没有别的按钮
        if ([self checkPathWithX1:x2 andX2:i+1 throughY:y2]&&[self
checkPathWithX1:i-1 andX2:x1 throughY:y1]&&[self checkPath
WithY1:y1-1 andY2:y2+1 throughX:i]) {
            return YES;
        }
    }
    } else if (i>x1){
        //判断有没有别的按钮
        if ([self checkPathWithX1:x1 andX2:Column throughY:y1]&&[self
checkPathWithX1:x2 andX2:Column throughY:y2]) {
            return YES;
        }
        //判断有没有别的按钮
        if ([self checkPathWithY1:y1-1 andY2:y2+1 throughX:i]&&[self
checkPathWithX1:x1 andX2:i+1 throughY:y1]&&[self checkPath
WithX1:x2 andX2:i+1 throughY:y2]) {
            return YES;
        }
    }
}

//再进行纵向判断
for (int i = 0; i<Row; i++) {
    if (i<y1) {
        //判断有没有别的按钮
        if ([self checkPathWithY1:-1 andY2:y1 throughX:x1]&&[self
checkPathWithY1:-1 andY2:y2 throughX:x2]) {
            return YES;
        }
    }
    //判断有没有别的按钮
    if ([self checkPathWithY1:i-1 andY2:y2 throughX:x2]&&[self
checkPathWithY1:i-1 andY2:y1 throughX:x1]&&[self checkPath
WithX1:x2-1 andX2:x1+1 throughY:i]) {

```

```

        return YES;
    }
} else if (i > y1 && i < y2) {
    //判断有没有别的按钮
    if ([self checkPathWithY1:i-1 andY2:y2 throughX:x2] && [self
        checkPathWithY1:y1 andY2:i+1 throughX:x1] && [self checkPath
        WithX1:x1-1 andX2:x2+1 throughY:i]) {
        return YES;
    }
} else if (i > y2) {
    //判断有没有别的按钮
    if ([self checkPathWithY1:y1 andY2:Row throughX:x1] && [self
        checkPathWithY1:y2 andY2:Row throughX:x2]) {
        return YES;
    }
    //判断有没有别的按钮
    if ([self checkPathWithY1:y2 andY2:i+1 throughX:x2] && [self
        checkPathWithY1:y1 andY2:i+1 throughX:x1] && [self checkPath
        WithX1:x1-1 andX2:x2+1 throughY:i]) {
        return YES;
    }
}
}
}
}

```

buttonClicked:方法用来触发当点击到动物按钮时的方法。程序代码如下:

```

-(void)buttonClicked:(UIButton *)sender
{
    //判断是否是第一次点击
    if (_currentButton == nil) {
        //如果是第一次点击
        _currentButton = sender;
        _currentButton.backgroundColor = [UIColor colorWithHue:0.1
            saturation:1.0 brightness:1.0 alpha:1.0]; //设置背景颜色
    } else {
        //不是第一次点击
        _lastButton = _currentButton;
        _currentButton = sender;
        //判断这次跟上次点击的是不是同一种水果
        if ([_currentButton.currentBackgroundImage isEqual:_lastButton.current
            BackgroundImage]) {
            //如果是,根据连连看的规则,判断两者之间是不是能走通
            UIButton *firstButton;
            UIButton *secondButton;
            if (_lastButton.tag < _currentButton.tag) {
                firstButton = _lastButton;
                secondButton = _currentButton;
            } else {
                firstButton = _currentButton;
                secondButton = _lastButton;
            }
            if ([self canPassFromLastButton:firstButton ToCurrentButton:
                secondButton]) {
                //在这里要根据 tag 值得到两个点击点的坐标
                int x1 = [self getXFromButtonTag:_lastButton.tag];
                int y1 = [self getYFromButtonTag:_lastButton.tag];
                int x2 = [self getXFromButtonTag:_currentButton.tag];
                int y2 = [self getYFromButtonTag:_currentButton.tag];
                map[y1][x1] = 0;
            }
        }
    }
}

```

```

        map[y2][x2] = 0;
        //移除指定的按钮
        [_lastButton removeFromSuperview];
        [_currentButton removeFromSuperview];
        _lastButton = nil;
        _currentButton = nil;
    }else {
        _lastButton.backgroundColor = [UIColor clearColor];
        //设置按钮的背景颜色
        _lastButton = nil;
        _currentButton.backgroundColor = [UIColor colorWithHue:0.1
        saturation:1.0 brightness:1.0 alpha:1.0];
    }
}
else{
    _lastButton.backgroundColor = [UIColor clearColor];
    //设置按钮的背景颜色
    _lastButton = nil;
    _currentButton.backgroundColor = [UIColor colorWithHue:0.1
    saturation:1.0 brightness:1.0 alpha:1.0];
    //设置按钮的背景颜色
}
}
}
}

```

loadView 方法实现视图的加载,即将存放图像的按钮添加到当前视图中。

```

- (void)loadView
{
    UIView *aView = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 320, 460)];
    //设置背景颜色
    [aView setBackgroundColor:[UIColor colorWithHue:0.3 saturation:0.5
    brightness:1.0 alpha:1.0]];
    self.view = aView;
    UIButton *back = [UIButton buttonWithType:UIButtonTypeCustom];
    back.frame=CGRectMake(260, 548, 50, 18); //设置框架
    [back setTitle:@"back" forState:UIControlStateNormal];
    back.titleLabel.font=[UIFont systemFontOfSize:20]; //设置字体
    [back setBackgroundColor:[UIColor blackColor]];
    [back addTarget:self action:@selector(anniu:) forControlEvents:UIControlEventTouchUpInside];
    [self.view addSubview:back]; //添加
    //所有的动物要用按钮来表示,一些上按钮的创建以及添加
    for (int i = 0; i < Row ; i++) {
        for (int j = 0; j < Colum ; j++) {
            map[i][j] = 1; //初始化数组,刚开始为1,如果点击消失后,都变成0;
            UIButton *animalButton = [UIButton buttonWithType:UIButtonTypeCustom];
            animalButton.frame = CGRectMake(10+j*Width, 10+i*Width, Width,
            Width);
            animalButton.tag = i*Colum + j+1; //设置 tag 值
            [self.view addSubview:animalButton];
            [animalButton addTarget:self action:@selector(buttonClicked:)
            forControlEvents:UIControlEventTouchUpInside]; //添加动作
        }
    }
    int randomNumber ;
    int animal[8]; //默认情况下初始值都会是 0
    for (int i = 0; i < 8; i++) {
        animal[i] = 0;
    }
}

```

```

    }
    //下面给每个按钮设置图像
    for (int i = 1; i<=Row*Colum; i++) {
        randomNumber = arc4random()%8; //生成随机数
        animal[randomNumber] ++;
        if (animal[randomNumber]<7) {
            [((UIButton *)[self.view viewWithTag:i]) setBackgroundImage:
             [UIImage imageNamed:[NSString stringWithFormat:@"%bg%d",random
             Number]] forState:UIControlStateNormal];
        }else{
            i--; //如果这轮产生的水果超出了原定的数量则还要重复这次的工作
        }
    }
    _lastButton = nil;
}

```

anniu:方法实现单击 back 按钮后界面的切换功能。程序代码如下:

```

- (IBAction)anniu: (id) sender{
    CATransition *transion=[CATransition animation];
    transion.type=kCATransitionMoveIn; //设置动画类型
    transion.duration=2.0; //设置动画的持续时间
    set=[self.storyboard instantiateViewControllerWithIdentifier:@"aaa"];
    [self.view addSubview:set.view];
    [self.view.layer addAnimation:transion forKey:@""];
}

```

在本实例中，连连看的游戏分为两大关卡：简单关和困难关。它们的代码大致相同，只是动图的图片在困难关需有所增多。由于篇幅的限制，请读者自行查看相关的源代码。

【代码解析】

本实例关键功能是路径的判断。下面就是这个知识点的详细讲解。

在所有的连连看中，路径的判断都是非常重要的。在本实例中，对于路径的判断首先需要获取选中的前一个点（即图像按钮）和当前的点（即图像按钮），代码如下：

```

//获取选中的前一个点的坐标
int x1 = [self getXFromButtonTag:lastButton.tag];
int y1 = [self getYFromButtonTag:lastButton.tag];
//获取选中的当前点的坐标
int x2 = [self getXFromButtonTag:currentButton.tag];
int y2 = [self getYFromButtonTag:currentButton.tag];

```

然后通过坐标对路径进行判断，代码如下：

```

if (x1==x2&&yl==y2) { //同一个点当然不能消掉
    .....
}else if (y1 == y2) { //判断是不是同一行
    .....
}
else if (x1 == x2) { //判断是不是同一列
    .....
}
else { //既不同一行也不是同一列，这样就得分为一个拐点跟两个拐点来考虑
    .....
}

```


实例36 人脸识别

【实例描述】

在不少的应用程序中多会进行人脸的识别。本实例就为各位读者实现此功能。首先,需要单击“换人”按钮,选择要进行识别的人脸。再点击“识别”按钮,进行人脸识别,如果识别成功,就对脸、眼睛和嘴巴进行标记;如果识别,就会弹出一个标题为“失败”的警告视图,告诉运行识别失败,运行效果如图2.25所示。

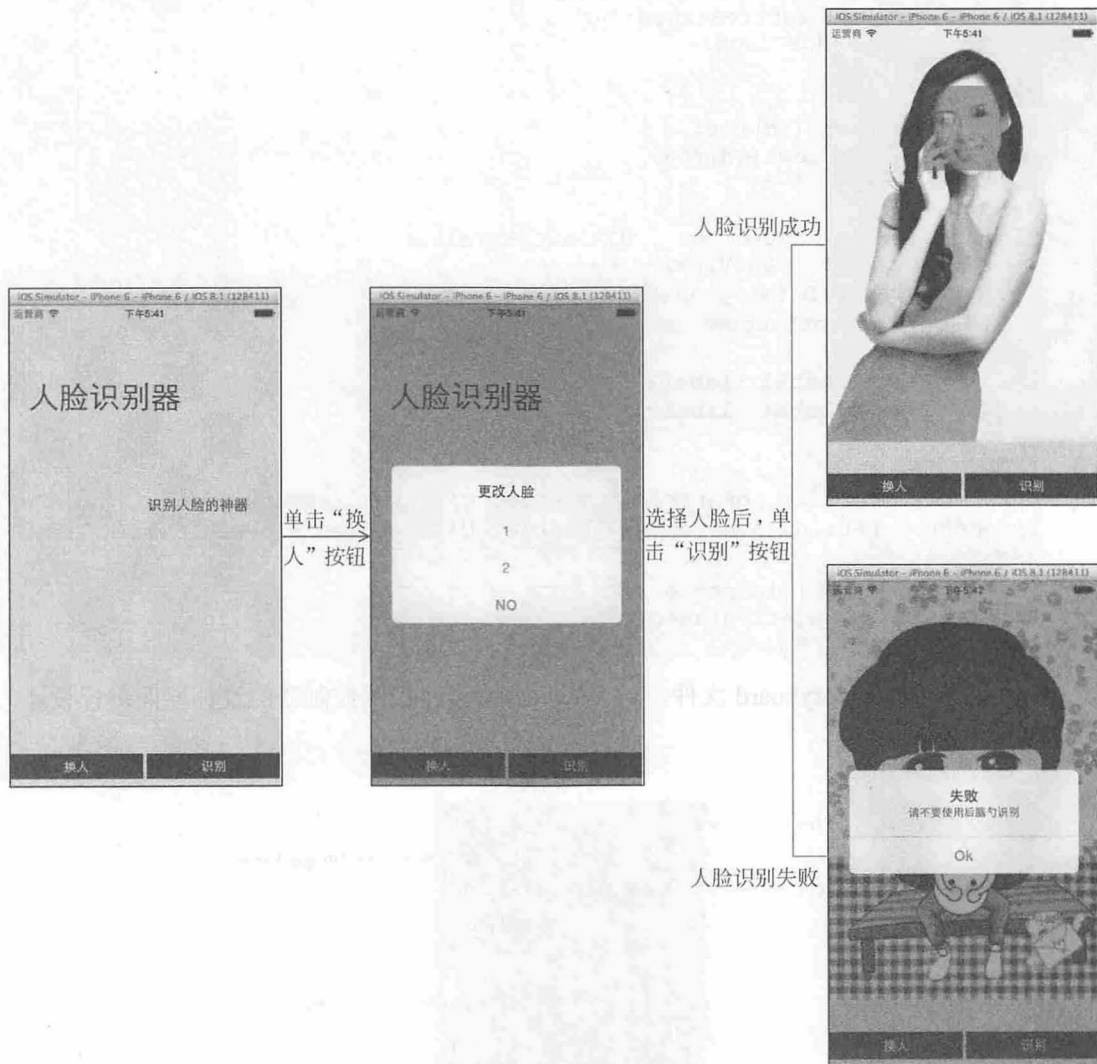


图 2.25 运行效果

【实现过程】

- (1) 创建一个项目,命名为“人脸识别”。
- (2) 添加图像 1.jpg、2.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件,编写代码,实现定义数据结构、基于 UIView 类的 UIProgressHUD 类的声明,以及对对象、插座变量、属性、动作的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
//UIProgressHUD 类的声明
@interface UIProgressHUD : UIView {
    //对象
    UILabel *_progressMessage;
    UIImageView *_doneView;
    UIWindow *_parentWindow;
    //定义数据结构
    struct {
        unsigned int isShowing:1;
        unsigned int isShowingText:1;
        unsigned int fixedFrame:1;
        unsigned int reserved:30;
    } _progressHUDFlags;
}
//方法
- (void)setText:(id)fp8;
- (void)showInView:(id)fp8;
- (void)hide;
@end
@interface ViewController : UIViewController{
    UIImageView* imageView;
    UIProgressHUD *progressHUD;
    CGRect rectFaceDetect;
    //插座变量
    IBOutlet UILabel *label1;
    IBOutlet UILabel *label2;
}
//属性
@property (retain, nonatomic) IBOutlet UIView *viewShow;
@property (retain, nonatomic) IBOutlet UIImageView *imageView;
//动作
- (IBAction)show:(id)sender;
- (IBAction)Change:(id)sender;
@end

```

(4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 2.26 所示。

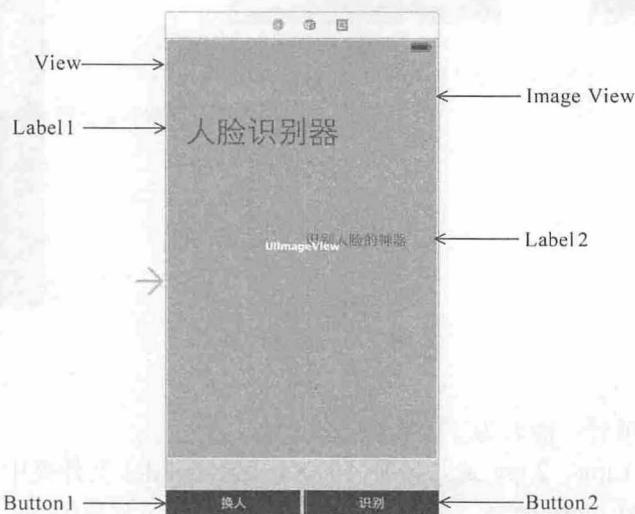


图 2.26 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-24 所示。

表 2-24 视图、控件设置

视图、控件	属 性 设 置	其 他
View		与插座变量viewShow关联
Image View		与插座变量imageView关联
Label1	Text: 人脸识别器 Font: System 36.0 Alignment: 居中	与插座变量label1关联
Label2	Text: 识别人脸的神器 Alignment: 居中	与插座变量label2关联
Button1	Title: 换人 Background: 黑色	与动作Change:关联
Button2	Title: 识别 Background: 黑色	与动作show:关联

(5) 打开 ViewController.m 文件, 编写代码, 实现选择人脸以及识别人脸的功能。使用的方法如表 2-25 所示。

表 2-25 ViewController.m文件中方法总结

方 法	功 能
setShowViewFrame	设置显示的视图框架
dealImageWhenItChanged	处理图像
removeAllMarkViews	移除所有的标记视图
faceDetect:	人脸检测
markAfterFaceDetect:	人脸标识
scaleImage:toScale:	缩放图像
showProgressIndicator:	显示指示器
hideProgressIndicator	隐藏指示器
show:	单击按钮显示指示器
Change:	弹出警告视图
alertView:clickedButtonAtIndex:	警告视图的响应

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, 人脸的选择需要使用 Change:和 alertView:clickedButtonAtIndex:方法实现。程序代码如下:

```

- (IBAction)Change:(id)sender {
    UIAlertView *a=[[UIAlertView alloc] initWithTitle:@"更改人脸" message:nil
    delegate:self cancelButtonTitle:@"NO" otherButtonTitles:@"1",@"2",
    nil];
    [a show];
}
//警告视图的响应
-(void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *b=[alertView buttonTitleAtIndex:buttonIndex];
    //判断是否选择了 Delete 按钮
    if([b isEqualToString:@"1"]){
        imageView.image = [UIImage imageNamed:@"1.jpg"];
    }
}

```

```

label1.hidden=YES;
label2.hidden=YES;
[self removeAllMarkViews];
}else{
    imageView.image = [UIImage imageNamed:@"2.jpg"];
    //设置图像视图显示的图像
    label1.hidden=YES;
    label2.hidden=YES;
    [self removeAllMarkViews];
}
}
}

```

单击 show 按钮后，首先显示一个加载的指示器。需要使用方法 show:、showProgressIndicator:、hideProgressIndicator 方法。其中 showProgressIndicator:方法实现指示器的显示。程序代码如下：

```

- (void)showProgressIndicator:(NSString *)text {
    self.view.userInteractionEnabled = FALSE;
    if(!progressHUD) {
        CGFloat w = 160.0f, h = 120.0f;
        //实例化对象
        progressHUD = [[UIProgressHUD alloc] initWithFrame:CGRectMake(
            (self.view.frame.size.width-w)/2, (self.view.frame.size.height-h)/2,
            w, h)];
        [progressHUD showInView:self.view]; //显示
    }
    [self performSelector:@selector(dealImageWhenItChanged) withObject:self
    afterDelay:5];
}

```

实现人脸的识别，需要使用 setShowViewFrame、dealImageWhenItChanged、removeAllMarkViews、faceDetect:、markAfterFaceDetect:、scaleImage:toScale:方法。其中，setShowViewFrame 方法实现对显示视图框架的设置。程序代码如下：

```

- (void)setShowViewFrame
{
    CGFloat scale = 1.;
    CGSize imgSize = imageView.image.size;
    CGRect vFrame = self.view.frame; //获取 view 的框架
    CGRect sFrame = viewShow.frame; //获取 viewShow 的框架
    if (imgSize.width/CGRectGetWidth(vFrame) > imgSize.height/CGRectGetHeight(
    vFrame)) {
        sFrame.size.width = CGRectGetWidth(vFrame); //设置宽度
        sFrame.size.height = imgSize.height * CGRectGetWidth(vFrame)/
        imgSize.width; //设置高度
        scale = CGRectGetWidth(vFrame)/imgSize.width;
    }else{
        sFrame.size.height = CGRectGetHeight(vFrame); //设置宽度
        sFrame.size.width = imgSize.width * CGRectGetHeight(vFrame)/
        imgSize.height; //设置高度
        scale = CGRectGetHeight(vFrame)/imgSize.height;
    }
    sFrame.origin.x = (CGRectGetWidth(vFrame)-CGRectGetWidth(sFrame))/2.;
    sFrame.origin.y = vFrame.origin.y + (CGRectGetHeight(vFrame) -CGRectGet
    Height(sFrame))/2.;
    viewShow.frame = sFrame; //设置框架
}

```

```

//图片根据 imgV 的 frame 进行重新缩放生成
UIImage *newImg = [self scaleImage:imageView.image toScale:scale];
imageView.image = newImg;           //设置图像
}

```

dealImageWhenItChanged 方法实现图像的处理。程序代码如下:

```

-(void)dealImageWhenItChanged
{
    [self setShowViewFrame];
    rectFaceDetect = CGRectZero;
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_
        DEFAULT, 0), ^{
        [self faceDetect:imageView.image];           //调用 faceDetect:方法
    });
}

```

faceDetect:方法实现对人脸的检测。程序代码如下:

```

- (void)faceDetect:(UIImage *)aImage
{
    CIImage* image = [CIImage imageWithCGImage:aImage.CGImage];
    NSDictionary *opts = [NSDictionary dictionaryWithObject:CIDetectorAccuracy
        High forKey:CIDetectorAccuracy];           //创建字典对象
    CIDetector* detector = [CIDetector detectorOfType:CIDetectorTypeFace
        context:nil options:opts];
    NSArray* features = [detector featuresInImage:image]; //创建数组
    //判断 features 中的个数是否为 0
    if ([features count]==0) {
        NSLog(@">>>>> 人脸检测【失败】啦 ~!!!"); //输出
    }
    NSLog(@">>>>> 人脸检测【成功】~!!!>>>>> "); //输出
    dispatch_async(dispatch_get_main_queue(), ^{
        [self markAfterFaceDetect:features]; //调用 markAfterFaceDetect:方法
    });
}

```

markAfterFaceDetect:方法实现人脸的标记。程序代码如下:

```

-(void)markAfterFaceDetect:(NSArray *)features
{
    [self hideProgressIndicator];           //隐藏指示器
    [self setShowViewFrame];
    //判断 features 中的个数是否为 0
    if ([features count]==0) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"失败
            "message:@"请不要使用后脑勺识别"delegate:self cancelButtonTitle:@"Ok"
            otherButtonTitles:nil, nil];           //创建警告视图
        [alert show];
        return;
    }
    //遍历
    for (CIFaceFeature *f in features){
        //旋转 180, 仅 y
        CGRect aRect = f.bounds;
        aRect.origin.y = self.viewShow.bounds.size.height - aRect.size.
            height - aRect.origin.y; //self.bounds.size
        UIView *vv = [[UIView alloc] initWithFrame:aRect];
        vv.tag = 100;           //设置 tag 值
    }
}

```



```

[vv setTransform:CGAffineTransformMakeScale(1, -1)];
vv.backgroundColor = [UIColor redColor];           //设置背景颜色
vv.alpha = 0.6;                                   //设置透明度
[self.viewShow addSubview:vv];
rectFaceDetect = aRect;
if (f.hasLeftEyePosition){
    //标记左眼的位置
    UIView *vv = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10, 10)];
    vv.tag = 100;                                  //设置 tag 值
    //旋转 180, 仅 y
    CGPoint newCenter = f.leftEyePosition;          //获取左眼的位置
    newCenter.y = self.viewShow.bounds.size.height-newCenter.y;
    vv.center = newCenter;                          //设置中心位置
    vv.backgroundColor = [UIColor yellowColor];
    [vv setTransform:CGAffineTransformMakeScale(1, -1)];
    vv.alpha = 0.6;                                  //设置透明度
    [self.viewShow addSubview:vv];
}
if (f.hasRightEyePosition) {
    //标记右眼的位置
    UIView *vv = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10, 10)];
    vv.tag = 100;                                  //设置 tag 值
    //旋转 180, 仅 y
    CGPoint newCenter = f.rightEyePosition;          //获取右眼的位置
    newCenter.y = self.viewShow.bounds.size.height-newCenter.y;
    vv.center = newCenter;                          //设置中心位置
    vv.backgroundColor = [UIColor blueColor];
    [vv setTransform:CGAffineTransformMakeScale(1, -1)];
    vv.alpha = 0.6;                                  //设置透明度
    [self.viewShow addSubview:vv];
}
if (f.hasMouthPosition) {
    //标记嘴巴的位置
    UIView *vv = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10, 10)];
    vv.tag = 100;                                  //设置 tag 值
    //旋转 180, 仅 y
    CGPoint newCenter = f.mouthPosition;             //获取嘴巴的位置
    newCenter.y = self.viewShow.bounds.size.height-newCenter.y;
    vv.center = newCenter;                          //设置中心位置
    vv.backgroundColor = [UIColor greenColor];
    [vv setTransform:CGAffineTransformMakeScale(1, -1)];
    vv.alpha = 0.6;                                  //设置透明度
    [self.viewShow addSubview:vv];
}
}
}
}

```

scaleImage:toScale:方法实现图像的缩放。程序代码如下:

```

- (UIImage *) scaleImage:(UIImage *)image toScale:(float)scaleSize {
    if (image) {
        UIGraphicsBeginImageContext(CGSizeMake(image.size.width * scaleSize,
            image.size.height * scaleSize));          //创建位图的上下文
        [image drawInRect:CGRectMake(0, 0, image.size.width * scaleSize,
            image.size.height * scaleSize)];          //绘制
        UIImage *scaledImage = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();
    }
}

```

```

        return scaledImage;
    }
    return nil;
}

```

【代码解析】

本实例关键功能是人脸的识别，以及标记眼睛、嘴巴的位置。下面依次讲解这两个知识点。

1. 人脸的识别

早在 iOS 5 系统中，苹果公司就引进了对人脸的识别技术。其中 `CIDetector` 获取人脸中图像中的位置。在此代码中，就是使用了 `CIDetector` 对人脸进行了识别，代码如下：

```

CIDetector* detector = [CIDetector detectorOfType:CIDetectorTypeFace
context:nil options:opts];
NSArray* features = [detector featuresInImage:image];
if ([features count]==0) {
    NSLog(@">>>>> 人脸检测【失败】啦 ~!!! ");
}
NSLog(@">>>>> 人脸检测【成功】~!!! >>>>> ");

```

2. 眼睛、嘴巴的位置

要想获取眼睛、嘴巴的位置，可使用 `CIFaceFeature` 类来实现。其中，获取左眼的位置，需要使用 `leftEyePosition` 属性。其语法形式如下：

```
@property(readonly, assign) CGPoint leftEyePosition;
```

右眼位置的获取需要使用 `rightEyePosition` 属性，其语法形式如下：

```
@property(readonly, assign) CGPoint rightEyePosition;
```

在此代码就是使用了 `leftEyePosition` 和 `rightEyePosition` 属性获取了左右眼的位置从而进行了人脸中眼睛的标记，代码如下：

```

if (f.hasLeftEyePosition) {
    //标记左眼的位置
    UIView *vv = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10, 10)];
    .....
    CGPoint newCenter = f.leftEyePosition;
    newCenter.y = self.viewShow.bounds.size.height-newCenter.y;
    vv.center = newCenter;
    .....
}
if (f.hasRightEyePosition) {
    //标记右眼的位置
    UIView *vv = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10, 10)];
    .....
    CGPoint newCenter = f.rightEyePosition;
    newCenter.y = self.viewShow.bounds.size.height-newCenter.y;
    vv.center = newCenter;
    .....
}

```

对于嘴巴的位置获取需要使用 `mouthPosition` 属性。其语法形式如下：

```
@property(readonly, assign) CGPoint mouthPosition;
```

在此代码就是使用了 `mouthPosition` 属性获取了嘴巴的位置从而进行了人脸中嘴巴的标记，代码如下：

```
if (f.hasMouthPosition) {
    UIView *vv = [[UIView alloc] initWithFrame:CGRectMake(0, 0, 10, 10)];
    .....
    CGPoint newCenter = f.mouthPosition;
    newCenter.y = self.viewShow.bounds.size.height-newCenter.y;
    vv.center = newCenter;
    .....
}
```

实例 37 逐层刷新图像

【实例描述】

本实例实现的功能是在图像视图进行切换时，实现一个逐层刷新的功能。运行效果如图 2.27 所示。

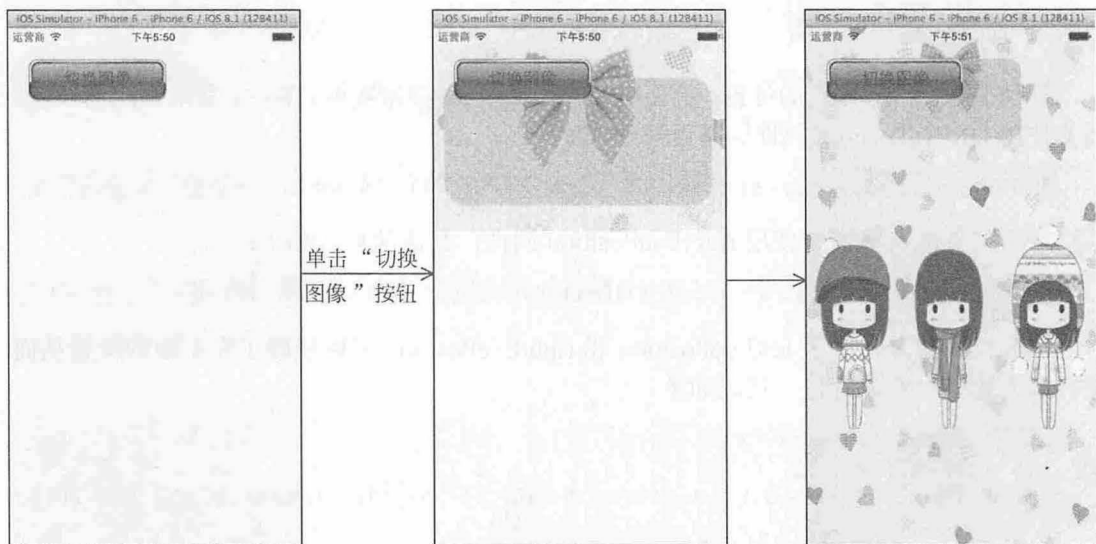


图 2.27 运行效果

【实现过程】

- (1) 创建一个项目，命名为“逐层刷新图像”。
- (2) 添加图像 1.jpg、2.jpg、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIImageView 类的 ShowImage 类。
- (4) 打开 ShowImage.h 文件，编写代码，实现实例变量、对象的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ShowImage : UIImageView{
    //实例变量
    int nowHeight;
    BOOL isAdding;
    //对象
```

```

NSTimer *timer;
CALayer *myLayer;
UIImage *img;
}
@end

```

(5) 打开 ShowImage.m 文件, 编写代码, 实现刷新功能。使用的方法如表 2-26 所示。

表 2-26 ShowImage.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
setImage:	设置图像
addFrameHeight	添加图像的高度

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。setImage:方法实现对图像的设置, 程序代码如下:

```

-(void)setImage:(UIImage *)image
{
    if (isAdding) {
        [timer invalidate];
        [myLayer removeFromSuperlayer]; //移除指定的图层
        myLayer = nil;
        isAdding = NO;
        nowHeight = 0;
        [super setImage:image]; //设置图像
    }
    img = image;
    CALayer *layer = [CALayer layer];
    //设置框架
    layer.frame = CGRectMake(self.frame.origin.x, self.frame.origin.y,
        self.frame.size.width, nowHeight);
    layer.contents= (id)image.CGImage;
    layer.masksToBounds =YES;
    layer.contentsGravity = kCAGravityBottom;
    myLayer = layer;
    [self.layer addSublayer:myLayer]; //添加图层对象
    timer = [NSTimer scheduledTimerWithTimeInterval:0.02 target:self
        selector:@selector(addFrameHeight) userInfo:nil repeats:YES];
    //创建时间定时器
    [timer fire];
    isAdding = YES;
}

```

addFrameHeight 方法实现图像的刷新功能, 即逐渐增加图像的高度。程序代码如下:

```

-(void)addFrameHeight
{
    if (self.superview) {
        nowHeight+=5;
        myLayer.frame = CGRectMake(self.frame.origin.x, self.frame.origin.y,
            self.frame.size.width, nowHeight); //设置框架
        if (nowHeight>= self.frame.size.height) {
            [timer invalidate];
            [super setImage:img]; //设置图层
            [myLayer removeFromSuperlayer]; //移除指定的图层对象
            myLayer = nil;
        }
    }
}

```

```
        nowHeight=0;
        isAdding =NO;
    }
}
```

（6）打开 ViewController.h 文件，编写代码，实现头文件、对象、插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "ShowImage.h"
@interface ViewController : UIViewController{
    UIImage *img;
    IBOutlet ShowImage *showImg;           //插座变量
}
- (IBAction)change: (id) sender;         //动作
@end
```

（7）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 2.28 所示。

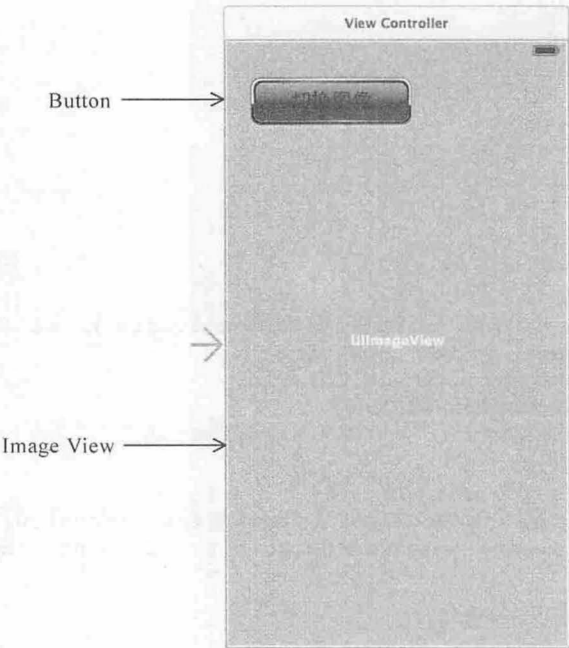


图 2.28 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 2-27 所示。

表 2-27 视图、控件设置

视图、控件	属 性 设 置	其 他
Button	Title: 切换图像 Font: 19.0 Text Color: 黑色 Background: 3.png Tag: 2	与动作change:关联

续表

视图、控件	属性设置	其他
Image View		Class: ShowImage 与插座变量showImg关联

(8) 打开 ViewController.m 文件, 编写代码, 实现单击按钮后图像的切换功能。程序代码如下:

```
- (IBAction)change:(UIButton*)sender {
    if (sender.tag == 1) {
        showImg.image = [UIImage imageNamed:@"1.jpg"];    //设置图像
        sender.tag = 2;
    }
    else
    {
        sender.tag = 1;
        showImg.image = [UIImage imageNamed:@"2.jpg"];    //设置图像
    }
}
```

【代码解析】

本实例关键功能是图像的逐层刷新效果。下面就是这个知识点的详细讲解。

在本实例中图像的逐层刷新效果的实现, 需要创建一个时间定时器, 代码如下:

```
timer = [NSTimer scheduledTimerWithTimeInterval:0.02 target:self selector:
@selector(addFrameHeight) userInfo:nil repeats:YES];
```

在每隔 0.02 秒后自定调用一次 addFrameHeight 方法, 它是逐层刷新效果的核心, 代码如下:

```
nowHeight+=5;
myLayer.frame = CGRectMake(self.frame.origin.x, self.frame.origin.y,
self.frame.size.width, nowHeight);
```

在每调用一次 addFrameHeight 方法实现, 它的高度都会增加 5。

实例 38 涂 鸦

【实例描述】

涂鸦就是为图像中添加一些关于自己的风格。本实例实现的功能是实现一个涂鸦效果。当用户选择蓝、黑、红中的某一视图后, 画出的颜色效果就会是相应的颜色。当选择橡皮擦时, 就会擦除想要去掉的部分; 如果将要删除全部绘制的内容, 可以直接选择撤销按钮。运行效果如图 2.29 所示。

【实现过程】

- (1) 创建一个项目, 命名为“涂鸦”。
- (2) 添加图像 1.jpg、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 Color 类。

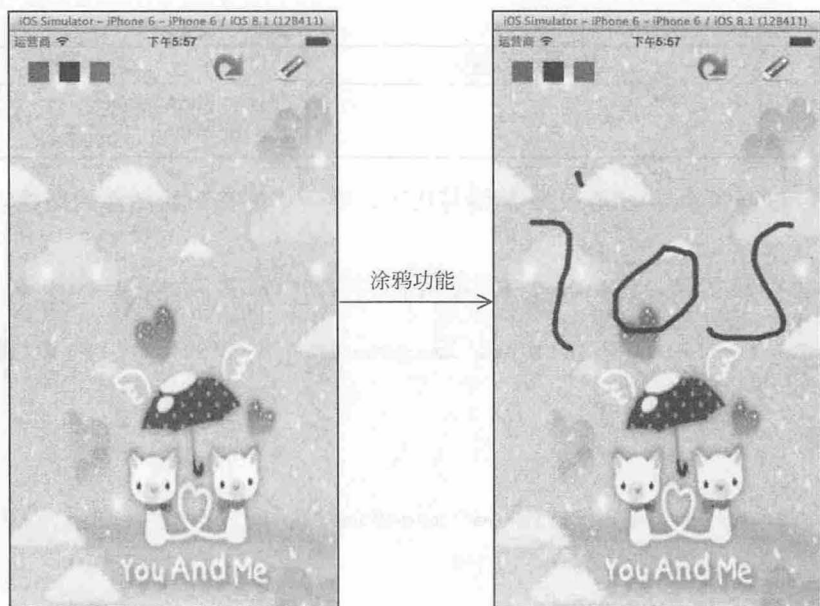


图 2.29 运行效果

(4) 打开 Color.h 文件，编写代码，实现对象以及属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface Color : UIView{
    NSArray *_array;                                //声明对象
}
@property(nonatomic,assign) NSInteger title;        //属性
@end
```

(5) 打开 Color.m 文件，编写代码，实现对视图的初始化以及触摸。程序代码如下：

```
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        _array = [[NSArray alloc] init];            //实例化对象
    }
    return self;
}
//开始触摸
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    _array = self.superview.subviews;
    //遍历数组_array
    for (id objtct in _array) {
        Color *colorView = objtct;
        //判断 colorView 的 title 属性是否为 1
        if (colorView.title == 1) {
            colorView.title = 0;
        }
    }
    self.title = 1;
}
```

(6) 创建一个基于 UIImageView 类的 DrawImage 类。

(7) 打开 DrawImage.h 文件, 编写代码, 实现头文件、实例变量、对象以及属性的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import "Color.h"
@interface DrawImage : UIImageView{
    //实例变量
    CGPoint _current;
    CGPoint _previous;
    //对象
    NSInteger _size ;
    NSArray * _array;
    UIColor * _previousColor;
}
@property(nonatomic,retain) UIColor *color;           //属性
@end
```

(8) 打开 DrawImage.m 文件, 编写代码, 实现对涂鸦界面的绘制以及涂鸦的功能。使用的方法如表 2-28 所示。

表 2-28 DrawImage.m 文件中方法总结

方 法	功 能
initWithFrame:	初始化
createColorView	创建涂鸦界面
eraser	橡皮擦的功能
cancel	撤销的功能
touchesBegan:withEvent:	开始触摸
touchesMoved:withEvent:	移动触摸
touchesEnded:withEvent:	结束触摸
drawImage	绘制

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。涂鸦界面的绘制需要使用 initWithFrame: 以及 createColorView 方法。其中, createColorView 方法实现对涂鸦界面的创建。程序代码如下:

```
- (void)createColorView {
    //创建一个背景色为蓝色的视图对象
    Color *view1 = [[Color alloc] initWithFrame:CGRectMake(20, 30, 20, 20)];
    view1.backgroundColor = [UIColor blueColor];
    [self addSubview:view1];
    //创建一个背景色为黑色的视图对象
    Color *view2 = [[Color alloc] initWithFrame:CGRectMake(50, 30, 20, 20)];
    view2.backgroundColor = [UIColor blackColor];
    [self addSubview:view2];
    //创建一个背景色为红色的视图对象
    Color *view3 = [[Color alloc] initWithFrame:CGRectMake(80, 30, 20, 20)];
    view3.backgroundColor = [UIColor redColor];
    [self addSubview:view3];
    //创建一个按钮对象, 实现撤销的功能
    UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    button.frame = CGRectMake(200, 20, 30, 30);
    [button setBackgroundImage:[UIImage imageNamed:@"2.png"] forState:
    UIControlStateNormal];
    [button addTarget:self action:@selector(cancel) forControlEvents:
```

```

UIControlEventTouchUpInside];
[self addSubview:button];
//创建一个按钮对象，实现橡皮擦的功能
UIButton *button1 = [UIButton buttonWithType:UIButtonTypeRoundedRect];
button1.frame = CGRectMake(260, 20, 35, 30);
[button1 setBackgroundImage:[UIImage imageNamed:@"3.png"] forState:UIControlStateNormal];
[button1 addTarget:self action:@selector(eraser) forControlEvents:UIControlEventTouchUpInside];
[self addSubview:button1];
}

```

涂鸦的实现需要使用 `touchesBegan:withEvent:`、`touchesMoved:withEvent:`、`touchesEnded:withEvent:`、`drawImage` 方法。其中，`touchesBegan:withEvent:`、`touchesMoved:withEvent:`、`touchesEnded:withEvent:` 方法实现触摸功能。程序代码如下：

```

//开始触摸
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];
    _previous = [touch locationInView:self.superview]; //获取触摸点
    _array = self.subviews;
    //遍历数组_array
    for (id object in _array) {
        Color *view = object;
        if (view.title == 1 && ![self.color isEqual:[UIColor clearColor]])
        {
            self.color = view.backgroundColor; //设置颜色
            _previousColor = self.color; //设置前一种颜色
        }
    }
}

//移动触摸
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    UITouch *touch = [touches anyObject];
    _current = [touch locationInView:self.superview]; //获取触摸点
    [self drawImage]; //绘制
    _previous = _current;
}

//结束触摸
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    _size = 5;
    self.color = _previousColor;
    //遍历_array 数组
    for (id object in _array) {
        Color *view = object;
        view.title = 0;
    }
}

```

`drawImage` 方法实现实现涂鸦的绘制。程序代码如下：

```

- (void)drawImage {
    UIGraphicsBeginImageContext(self.frame.size); //创建位图上下文
    CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
    [self.image drawInRect:CGRectMake(0, 0, self.bounds.size.width,
    self.bounds.size.height)]; //绘制
}

```

```

CGContextSetBlendMode(context, kCGBlendModeCopy);
CGContextSetLineCap(context, kCGLineCapRound);
CGContextSetLineWidth(context, _size); //设置线宽
CGContextSetStrokeColorWithColor(context, self.color.CGColor);
CGContextBeginPath(context);
CGContextMoveToPoint(context, _previous.x, _previous.y); //绘制开始点
CGContextAddLineToPoint(context, _current.x, _current.y); //绘制结束点
CGContextStrokePath(context); //绘制
self.image = UIGraphicsGetImageFromCurrentImageContext(); //设置图像
UIGraphicsEndImageContext();
}

```

(9) 打开 ViewController.h 文件, 编写代码, 实现头文件、对象、实例变量的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "DrawImage.h"
@interface ViewController : UIViewController{
    UIImageView *_imageView;
    NSInteger _flag;
    UIButton *_button;
}
@end

```

(10) 打开 ViewController.m 文件, 编写代码, 实现对界面的设计以及涂鸦。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    _imageView = [[UIImageView alloc] init];
    _imageView.frame = CGRectMake(0, 0, 320, 568);
    _imageView.image = [UIImage imageNamed:@"1.jpg"]; //设置图像
    [self.view addSubview:_imageView];
    DrawImage *drawView = [[DrawImage alloc] init];
    drawView.frame = CGRectMake(0, 0, 320, 568); //设置框架
    [self.view addSubview:drawView];
}

```

【代码解析】

本实例关键功能是涂鸦功能以及橡皮擦的功能。下面依次讲解这两个知识点。

1. 涂鸦功能

在本实例中涂鸦功能的实现, 是通过 CGContextMoveToPoint 函数和 CGContextAddLineToPoint 函数实现的, 代码如下:

```

CGContextMoveToPoint(context, _previous.x, _previous.y);
CGContextAddLineToPoint(context, _current.x, _current.y);

```

2. 橡皮擦功能

在实现橡皮擦的擦除功能时, 需要使用 CGContext 的 CGContextSetBlendMode 函数,

它的功能是设置混合模式。其语法形式如下：

```
void CGContextSetBlendMode (
    CGContextRef context,
    CGBlendMode mode
);
```

其中，CGContextRef context 表示上下文；CGBlendMode mode 表示混合模式。混合模式的共有 28 种，如表 2-29 所示。

表 2-29 混合模式

模 式	解 释
kCGBlendModeNormal	默认
kCGBlendModeMultiply	正片叠底混合模式，它是指定将前景的图片采样与背景图片采样相乘
kCGBlendModeScreen	屏幕混合模式，它是指定将前景图片采样的反转色与背景图片的反转色相乘
kCGBlendModeOverlay	叠加混合模式
kCGBlendModeDarken	暗化混合模式
kCGBlendModeLighten	亮化混合模式
kCGBlendModeColorDodge	色彩减淡模式
kCGBlendModeColorBurn	色彩加深模式
kCGBlendModeSoftLight	柔光混合模式
kCGBlendModeHardLight	强光混合模式
kCGBlendModeDifference	差值混合模式
kCGBlendModeExclusion	排除混合模式
kCGBlendModeHue	色相混合模式
kCGBlendModeSaturation	饱和度混合模式
kCGBlendModeColor	颜色混合模式
kCGBlendModeLuminosity	亮度混合模式
kCGBlendModeClear	清除混合模式
kCGBlendModeCopy	复制模式
kCGBlendModeSourceIn	来源中的混合模式
kCGBlendModeSourceOut	来源外的混合模式
kCGBlendModeSourceAtop	来源顶部的混合模式
kCGBlendModeDestinationOver	目标上的混合模式
kCGBlendModeDestinationIn	目标中的混合模式
kCGBlendModeDestinationOut	目标外的混合模式
kCGBlendModeDestinationAtop	目标顶部的混合模式
kCGBlendModeXOR	异或模式
kCGBlendModePlusDarker	加上深色混合模式
kCGBlendModePlusLighter	加上亮色混合模式

在此代码中就是将 CGContextSetBlendMode 函数中的混合模式设置为了 kCGBlendModeCopy，实现了橡皮擦的功能。代码如下：

```
CGContextSetBlendMode(context, kCGBlendModeCopy);
```

其中，context 表示上下文，kCGBlendModeCopy 表示复制模式。

实例 39 图像的 3D 效果浏览

【实例描述】

为了让用户在浏览图像时有 3D 的视觉效果,本实例就为读者实现一个 3D 效果浏览图像的功能。当用户拖动屏幕时,图像就是以 3D 效果进行旋转,从而实现图像的浏览。运行效果如图 2.30 所示。

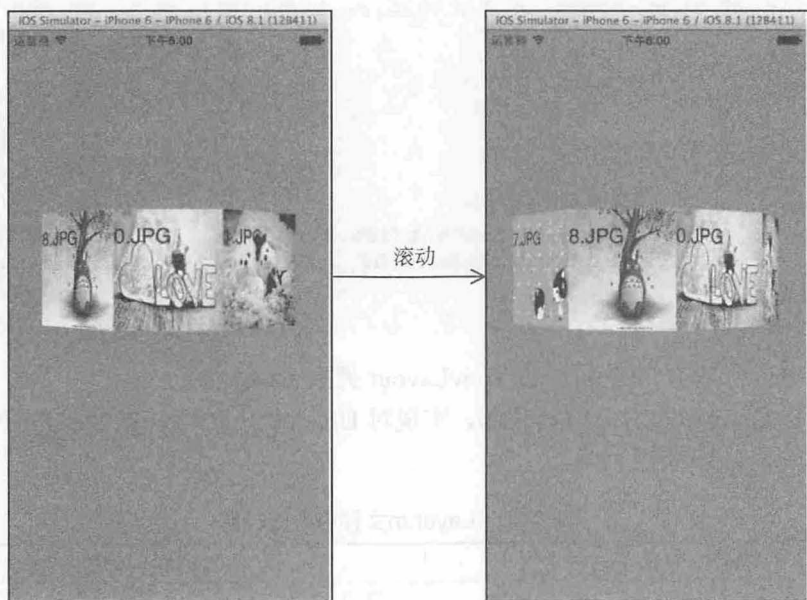


图 2.30 运行效果

【实现过程】

- (1) 创建一个项目,命名为“图像的 3D 效果浏览”。
- (2) 添加图像 0.jpg~8.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UICollectionViewCell 类的 ImageCell 类。
- (4) 打开 ImageCell.h 文件,编写代码,实现对象以及属性的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface ImageCell : UICollectionViewCell{
    //对象
    UIImageView *imageView ;
    UILabel *titleLabel;
}
//属性
@property (nonatomic,strong) UIImageView *imageView;
@property (nonatomic,strong) UILabel *titleLabel;
@end
```

- (5) 打开 ImageCell.m 文件,编写代码,实现 UICollectionView 集合视图中单元格的初始化以及布局。程序代码如下:

```

//初始化
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        //Initialization code
        //图像视图对象的创建以及设置
        imageView = [[UIImageView alloc] init];
        [self.contentView addSubview:imageView];
        //标签对象的创建以及设置
        titleLabel = [[UILabel alloc] init];
        titleLabel.textColor = [UIColor blackColor];
        [self.contentView addSubview:titleLabel];
    }
    return self;
}

//布局
- (void)layoutSubviews
{
    [super layoutSubviews];
    imageView.frame = self.contentView.bounds;           //设置图像视图的框架
    titleLabel.frame = CGRectMake(0.0f, 0.0f, self.contentView.bounds.size.
    width, 44.0f);                                       //设置标签的框架
}

```

(6) 创建一个基于 UICollectionViewLayout 类的 Layer 类。

(7) 打开 Layer.m 文件，编写代码，实现对 UICollectionView 集合视图的自定义布局。使用的方法如表 2-30 所示。

表 2-30 Layer.m 文件中方法总结

方 法	功 能
collectionViewContentSize	获取尺寸
shouldInvalidateLayoutForBoundsChange:	判断是否该刷新布局
layoutAttributesForItemAtIndexPath:	获取对应于索引路径的位置的单元格的布局属性
layoutAttributesForElementsInRect:	获取矩形区域中所有元素的布局属性

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，layoutAttributesForItemAtIndexPath: 方法实现对对应于索引路径的位置的单元格布局属性的获取。程序代码如下：

```

- (UICollectionViewLayoutAttributes *)layoutAttributesForItemAtIndexPath:
(NSIndexPath *)indexPath
{
    //3D 效果的代码
    UICollectionViewLayoutAttributes* attributes = attributes = [UICollectionView
    ViewLayoutAttributes layoutAttributesForCellWithIndexPath:indexPath];
    UICollectionView *collection = self.collectionView;
    float width = collection.frame.size.width;           //获取宽度
    float x = collection.contentOffset.x;
    CGFloat arc = M_PI * 2.0f;
    int numberOfVisibleItems = [self.collectionView numberOfItemsIn
    Section:0 ];
    attributes.center = CGPointMake(x+160, 240.0f);       //设置中心位置
    attributes.size = CGSizeMake(90.0f, 100.0f);         //设置尺寸
    CATransform3D transform = CATransform3DIdentity;
}

```

```

transform.m34 = -1.0f/700.0f;
CGFloat radius = attributes.size.width/2/ tanf(arc/2.0f/numberOfVisibleItems);
CGFloat angle = (indexPath.row-x/width+1)/ numberOfVisibleItems * arc;
transform = CATransform3DRotate(transform, angle, 0.0f, 1.0f, 0.0f);
//旋转
transform = CATransform3DTranslate(transform, 0.f, 0.0f, radius);
//平移
attributes.transform3D = transform ;
return attributes;
}

```

layoutAttributesForElementsInRect:方法实现对指定矩形区域中所有元素的布局属性的获取。程序代码如下:

```

- (NSArray*) layoutAttributesForElementsInRect: (CGRect) rect
{
    NSArray *arr = [super layoutAttributesForElementsInRect:rect];
    //判断数组 arr 中元素的个数
    if ([arr count] >0) {
        return arr;
    }
    NSMutableArray* attributes = [NSMutableArray array]; //创建数组
    //循环, 添加对象
    for (NSInteger i=0 ; i < [self.collectionView numberOfItemsInSection:0] ; i++) {
        NSIndexPath* indexPath = [NSIndexPath indexPathForItem:i inSection:0];
        [attributes addObject:[self layoutAttributesForItemAtIndexPath:
            indexPath]]; //添加对象
    }
    return attributes;
}

```

(8) 打开 ViewController.h 文件, 编写代码, 实现宏定义、头文件、对象以及遵守协议的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#define KCellIdentifier @"identifier"
#import "ImageCell.h"
#import "Layout.h"
@interface ViewController : UIViewController<UICollectionViewDataSource,
UICollectionViewDelegate>{
    UICollectionView *collectionView;
}
@end

```

(9) 打开 ViewController.m 文件, 编写代码, 实现 UICollectionView 集合视图的内容填充以及滚动效果。使用的方法如表 2-31 所示。

表 2-31 UICollectionView文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
numberOfSectionsInCollectionView:	获取块数
collectionView:numberOfItemsInSection:	定义展示的集合视图中单元格的个数
collectionView:cellForItemAtIndexPath:	获取每个单元格中展示的内容
scrollViewDidScroll:	滚动

其中，CollectionView 集合视图的内容填充需要使用 `viewDidLoad`、`numberOfSectionsInCollectionView`、`collectionView:numberOfItemsInSection` 和 `collectionView:cellForItemAtIndexPath` 方法。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    float width = self.view.frame.size.width;           //获取 view 的宽度
    float height = self.view.frame.size.height;         //获取 view 的高度
    //创建并设置集合视图对象
    collectionView = [[UICollectionView alloc] initWithFrame:CGRectMake(
    0.0f, 0.0f, width, height)collectionViewLayout:[UICollectionViewLayout alloc] init]];
    [collectionView registerClass:[ImageCell class] forCellWithReuseIdentifier:KCellIdentifier];
    collectionView.backgroundColor = [UIColor grayColor]; //设置背景颜色
    collectionView.delegate = self;
    [collectionView setContentOffset:CGPointMake(width, 0.0f)];
    collectionView.dataSource = self;                   //设置数据源
    [self.view addSubview:collectionView];
}
//获取块数
- (NSInteger)numberOfSectionsInCollectionView:(UICollectionView *)collectionView
{
    return 1;
}
//定义展示的单元格的个数
- (NSInteger)collectionView:(UICollectionView *)collectionView numberOfItemsInSection:(NSInteger)section
{
    return 9;
}
//获取每个单元格展示的内容
- (UICollectionViewCell *)collectionView:(UICollectionView *)cView cellForItemAtIndexPath:(NSIndexPath *)indexPath
{
    ImageCell *cell = (ImageCell *)[cView dequeueReusableCellWithIdentifierWithIdentifier:KCellIdentifierforIndexPath:indexPath];
    //判断 cell 是否为空
    if (!cell) {
        NSLog(@"what ? cell is nil ? It's joke !");
        return nil;
    }
    NSString *imageName = [NSString stringWithFormat:@"%d.JPG", indexPath.row];
    cell.imageView.image = [UIImage imageNamed:imageName]; //设置图像
    cell.titleLabel.text = imageName;                       //设置文本内容
    return cell;
}
```

`scrollViewDidScroll` 方法实现在滚动视图中的无限滚动。程序代码如下：


```

- (void)scrollViewDidScroll:(UIScrollView *)_scrollView
{
    //无限循环....
    float targetX = _scrollView.contentOffset.x;//获取可滚动区域偏移量的 x 值
    int numCount = [collectionView numberOfItemsInSection:0];
    float ITEM_WIDTH = _scrollView.frame.size.width;    //获取滚动视图的宽度
    if (numCount>=3)
    {
        if (targetX < ITEM_WIDTH/2) {
            //设置滚动视图的可滚动区域的偏移量
            _scrollView.contentOffset=CGPointMake(targetX+ITEM_WIDTH *numCount, 0);
        }
        else if (targetX >ITEM_WIDTH/2+ITEM_WIDTH *numCount)
        {
            //设置滚动视图的可滚动区域的偏移量
            _scrollView.contentOffset=CGPointMake(targetX-ITEM_WIDTH *numCount, 0);
        }
    }
}

```

【代码解析】

本实例关键功能是 3D 旋转效果以及滚动视图的无限滚动。下面依次讲解这两个知识点。

1. 3D旋转效果

3D 效果的实现一般采用采用的是 CATransform3D 动画。在本实例中 3D 效果的旋转需要使用 CATransform3DRotate 函数，其语法形式如下：

```

CATransform3D CATransform3DRotate (
    CATransform3D t,
    CGFloat angle,
    CGFloat x,
    CGFloat y,
    CGFloat z
)

```

其中，参数说明如下：

- ❑ CATransform3D t 表示 CATransform3D 类型的数据；
- ❑ CGFloat angle 表示旋转的弧度；
- ❑ CGFloat x 表示 x 的坐标；
- ❑ CGFloat y 表示 y 的坐标；
- ❑ CGFloat z 表示 z 的坐标。

在本实例中使用 CATransform3DRotate 函数的代码如下：

```
transform = CATransform3DRotate(transform, angle, 0.0f, 1.0f, 0.0f);
```

其中，参数说明如下：

- ❑ transform 表示 CATransform3D 类型的数据；

- ❑ angle 表示旋转的弧度;
- ❑ 0.0f 表示 x 的坐标;
- ❑ 1.0f 表示 y 的坐标;
- ❑ 0.0f 表示 z 的坐标。

2. 滚动视图的无限滚动

在本实例中滚动视图的无限滚动都是通过对 contentOffset 的属性的设置。代码如下:

```
if (numCount>=3)
{
    if (targetX < ITEM_WIDTH/2) {
        _scrollView.contentOffset=CGPointMake(targetX+ITEM_WIDTH*numCount, 0);
    }
    else if (targetX >ITEM_WIDTH/2+ITEM_WIDTH *numCount) {
        _scrollView.contentOffset=CGPointMake(targetX-ITEM_WIDTH*numCount,0);
    }
}
```

第3章 图 表

在人口普查等统计信息时，只有图或者表，往往很难直观地展示统计信息的属性，如时间性、数量性等。这时，引进了新的概念——图表。最常见的图表有饼状图、柱状图、折线图。本章将主要讲解关于图表的一些实例。

实例 40 饼 状 图

【实例描述】

饼状图是一个划分为几个扇形的圆形统计图表，用于描述量、频率或百分比之间的相对关系。在饼图中，每个扇区的弧长（以及圆心角和面积）大小为其所表示的数量的比例。本实例主要的功能就是绘制这么一个饼状图，用户通过第一个滑块控件，可以将饼状图变为一个圆环效果的图。用户通过第二个滑块控件，可以将饼状图中的扇形改变。运行效果如图 3.1 所示。

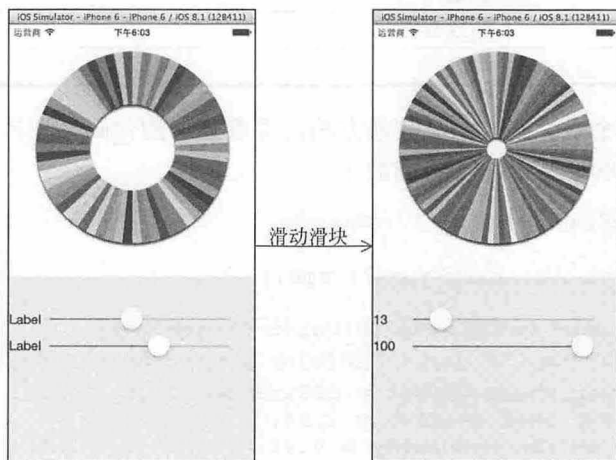


图 3.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“饼状图”。
- (2) 创建一个基于 UIView 类的 PieView 类。
- (3) 打开 PieView.h 文件，编写代码，实现头文件、协议、属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@class PieView;
```

```

// PieViewDelegate 协议
@protocol PieViewDelegate <NSObject>
- (CGFloat)centerCircleRadius;
@end

// PieViewDataSource 协议
@protocol PieViewDataSource <NSObject>
@required
- (int)numberOfSlicesInPieChartView:(PieView *)pieChartView; //获取切片个数
- (double)pieChartView:(PieView *)pieChartView valueForSliceAtIndex:(NSInteger)index; //获取切片的值
- (UIColor*)pieChartView:(PieView*)pieChartView colorForSliceAtIndex:(NSInteger)index; //获取切片的颜色
@optional
- (NSString*)pieChartView:(PieView*)pieChartView titleForSliceAtIndex:(NSInteger)index;
@end

@interface PieView : UIView
//属性
@property (nonatomic, assign) id <PieViewDataSource> datasource;
@property (nonatomic, assign) id <PieViewDelegate> delegate;
- (void)reloadData; //加载数据
@end

```

(4) 打开 PieView.m 文件，编写代码，实现饼状图的绘制。使用的方法如表 3-1 所示。

表 3-1 PieView.m 文件中方法总结

方 法	功 能
initWithFrame:	初始化
reloadData	加载数据
drawRect:	绘制

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，initWithFrame: 方法实现对饼状图的初始化。程序代码如下：

```

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor clearColor];
        self.layer.shadowColor = [[UIColor blackColor] CGColor]; //设置阴影颜色
        self.layer.shadowOffset = CGSizeMake(0.0f, 2.5f); //获取阴影的偏移量
        self.layer.shadowRadius = 1.9f;
        self.layer.shadowOpacity = 0.9f;
    }
    return self;
}

```

drawRect:方法实现了对饼状图的绘制。程序代码如下：

```

- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
    CGFloat theHalf = rect.size.width/2;
    CGFloat lineWidth = theHalf;
    //判断是否执行了 centerCircleRadius 方法
    if ([self.delegate respondsToSelector:@selector(centerCircleRadius)])

```

```

{
    lineWidth -= [self.delegate centerCircleRadius];
    NSAssert(lineWidth <= theHalf, @"wrong circle radius");
}
//为变量赋值
CGFloat radius = theHalf-lineWidth/2;
CGFloat centerX = theHalf;
CGFloat centerY = rect.size.height/2;
//绘制
double sum = 0.0f;
int slicesCount = [self.datasource numberOfSlicesInPieChartView:self];
//获取切片个数

//循环, 求和
for (int i = 0; i < slicesCount; i++)
{
    sum += [self.datasource pieChartView:self valueForSliceAtIndex:i];
}
float startAngle = - M_PI_2;
float endAngle = 0.0f;
//循环, 绘制切片
for (int i = 0; i < slicesCount; i++)
{
    double value = [self.datasource pieChartView:self valueForSlice
    AtIndex:i];
    endAngle = startAngle + M_PI*2*value/sum;
    CGContextAddArc(context, centerX, centerY, radius, startAngle, end
    Angle, false); //添加弧
    UIColor *drawColor = [self.datasource pieChartView:self colorFor
    Slice AtIndex:i];
    CGContextSetStrokeColorWithColor(context, drawColor.CGColor); //设置线的颜色
    CGContextSetLineWidth(context, lineWidth); //设置线的宽度
    CGContextStrokePath(context); //绘制
    startAngle += M_PI*2*value/sum;
}
}
}

```

(5) 打开 ViewController.h 文件, 编写代码, 实现头文件、插座变量、对象以及动作的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "PieView.h"
@interface ViewController : UIViewController<PieViewData Source, Pie View
Delegate>{
    //声明关于视图的插座变量
    IBOutlet UIView *vv;
    //声明关于滑块控件的插座变量
    IBOutlet UISlider *holeSlider;
    IBOutlet UISlider *slicesSlider;
    //声明关于标签的插座变量
    IBOutlet UILabel *holeLabel;
    IBOutlet UILabel *valueLabel;
    PieView *pieView;
}
- (IBAction)Change:(UISlider*)slider;
@end

```

(6) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计,

效果如图 3.2 所示。

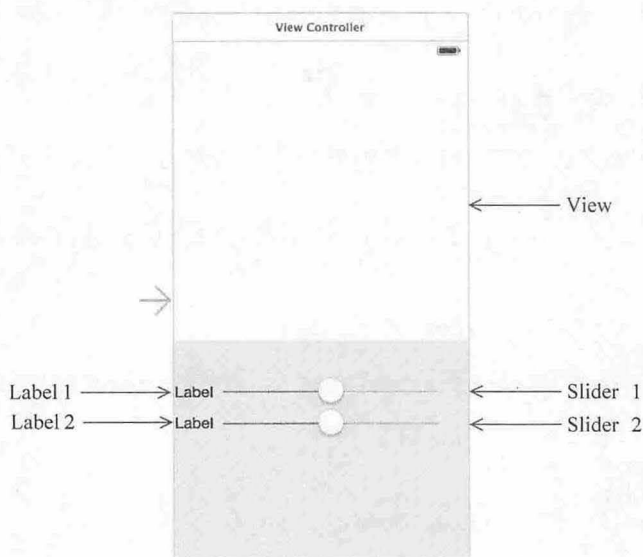


图 3.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 3-2 所示。

表 3-2 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	
View		与插座变量 vv 关联
Label1		与插座变量 holeLabel 关联
Label2		与插座变量 valueLabel 关联
Slider1		与插座变量 holeSlider 关联 与动作 Change:关联
Slider2		与插座变量 slicesSlider 关联 与动作 Change:关联

(7) 打开 ViewController.m 文件，编写代码，实现将绘制的饼状图添加到界面中，并实现对饼状图的控制。使用的方法如表 3-3 所示。

表 3-3 ViewController.m文件中方法总结

方 法	功 能
UIColor *GetRandomUIColor	获取颜色
viewDidLoad	视图加载后调用，实现初始化
centerCircleRadius	获取中心圆的半径
numberOfSlicesInPieChartView:	获取切片个数
pieChartView:colorForSliceAtIndex:	获取切片的颜色
pieChartView:valueForSliceAtIndex:	获取切片的值
Change:	滑动滑块

其中，UIColor *GetRandomUIColor 方法实现了对随机产生颜色的获取。程序代码如下：

```
static inline UIColor *GetRandomUIColor()
{
    //生成随机值
    CGFloat r = arc4random() % 255;
    CGFloat g = arc4random() % 255;
    CGFloat b = arc4random() % 255;
    UIColor * color = [UIColor colorWithRed:r/255 green:g/255 blue:b/255
        alpha:1.0f];
    return color;                //返回颜色
}
```

viewDidLoad 方法实现的功能是将绘制的具有饼状图的视图添加到界面中，以及滑块控件的初始化。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    pieView = [[PieView alloc] initWithFrame:CGRectMake(35.0f, 35.0f, 250.0f,
        250.0f)];
    pieView.delegate = self;                //设置协议
    pieView.datasource = self;            //设置委托
    [vv addSubview:pieView];
    holeSlider.tag = 7;                    //设置 tag 值
    holeSlider.minimumValue = 0.0f;
    holeSlider.maximumValue = 250/2 - 1;    //设置最大值
    int max = holeSlider.maximumValue;
    holeSlider.value = arc4random() % max;
    slicesSlider.tag = 77;                //设置 tag 值
    slicesSlider.minimumValue = 0.0f;        //设置最小值
    slicesSlider.maximumValue = 100.0f;
    slicesSlider.value = arc4random() % 100; //设置当前的值
}
```

Change:方法实现了当滑动滑块控件中的滑块时，将饼状图的形状进行重新设置以及在标签中显示滑块当前的值。程序代码如下：

```
-(IBAction)Change:(UISlider *)slider{
    if (slider.tag == 77)
        valueLabel.text = [NSString stringWithFormat:@"%f", slider.value];
        //设置标签内容
    if (slider.tag == 7)
        holeLabel.text = [NSString stringWithFormat:@"%f", slider.value];
        //设置标签内容
    [pieView reloadData];
}
```

【代码解析】

本实例关键功能是饼状图的绘制。下面就是这个知识点的详细讲解。

饼状图的绘制就是在圆形图中创建一些曲线段，需要使用 **CGContext** 的 **CGContextAddArc** 函数实现。其语法形式如下：

```
void CGContextAddArc (
    CGContextRef c,
    CGFloat x,
    CGFloat y,
    CGFloat radius,
    CGFloat startAngle,
    CGFloat endAngle,
    int clockwise
);
```

其中，参数说明如下：

- ❑ CGContextRef c 表示上下文；
- ❑ CGFloat x 表示圆点 x 坐标值；
- ❑ CGFloat y 表示圆点 y 坐标值；
- ❑ CGFloat radius 表示半径；
- ❑ CGFloat startAngle 表示开始的弧度；
- ❑ CGFloat endAngle 表示结束的弧度；
- ❑ int clockwise 表示绘制的方向（0 为顺时针，1 为逆时针）。

在此代码中就使用了 CGContextAddArc 函数绘制了一个饼状图，代码如下：

```
CGContextAddArc(
    context,
    centerX,
    centerY,
    radius,
    startAngle,
    endAngle,
    .false
);
```

其中，参数说明如下：

- ❑ context 表示上下文；
- ❑ centerX 表示圆点 x 坐标值；
- ❑ centerY 表示圆点 y 坐标值；
- ❑ radius 表示半径；
- ❑ startAngle 表示开始的弧度；
- ❑ endAngle 表示结束的弧度；
- ❑ false 表示绘制的方向，它是顺时针绘制的。

实例 41 柱 状 图

【实例描述】

柱状图是一种以长方形的长度为变量的统计图表，它主要用来比较两个或者两个以上的数值。本实例就为读者实现一个柱状图的效果。当用户单击界面的 Change 按钮后，柱

状图将重新绘制。运行效果如图 3.3 所示。

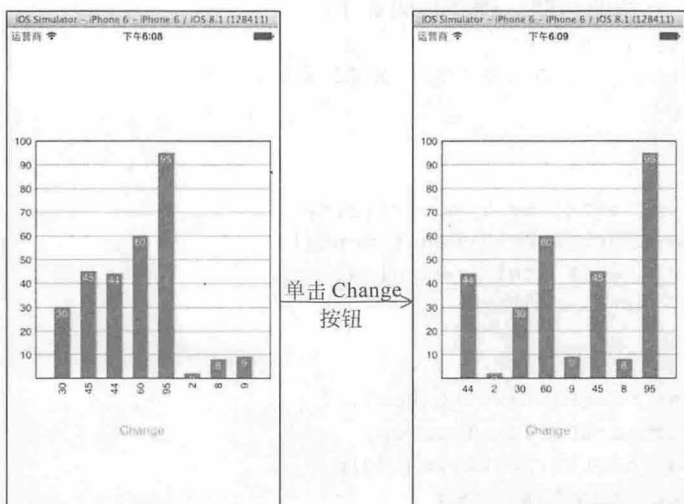


图 3.3 运行效果

【实现过程】

- (1) 创建项目，命名为“柱状图”。
- (2) 创建一个基于 NSMutableArray 类的分类 MutableArray。
- (3) 打开 NSMutableArray+MutableArray.h 文件，编写代码，实现方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
@interface NSMutableArray (MutableArray)
- (void)shuffle;
@end
```

- (4) 打开 NSMutableArray+MutableArray.m 文件，编写代码，对 shuffle 方法进行定义，实现随机元素的交换。程序代码如下：

```
#import "NSMutableArray+MutableArray.h"
@implementation NSMutableArray (MutableArray)
- (void)shuffle
{
    NSUInteger count = [self count];
    //循环，交换对象
    for (NSUInteger i = 0; i < count; ++i)
    {
        NSUInteger nElements = count - i;
        NSUInteger n = arc4random_uniform(nElements) + i;
        [self exchangeObjectAtIndex:i withObjectAtIndex:n]; //交换对象
    }
}
@end
```

- (5) 创建一个基于 UIView 类的 BarView 类。

(6) 打开 BarView.h 文件，编写代码，实现宏定义、数据结构的定义、协议、实例变量、对象、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#define DEGREES_TO_RADIANS(x) (M_PI * x / 180.0) //宏定义
//数据结构的定义
typedef enum
{
    SimpleBarChartXLabelTypeVerticle,
    SimpleBarChartXLabelTypeHorizontal,
    SimpleBarChartXLabelTypeAngled
} SimpleBarChartXLabelType;
typedef enum
{
    SimpleBarChartBarTextTypeRoof,
    SimpleBarChartBarTextTypeTop,
    SimpleBarChartBarTextTypeMiddle
} SimpleBarChartBarTextType;
@class BarView;
//BarViewDataSource 协议
@protocol BarViewDataSource <NSObject>
@required
- (NSInteger)numberOfBarsInBarChart:(BarView *)barChart;
    //获取柱形的个数
- (CGFloat)barChart:(BarView *)barChart valueForBarAtIndex: (NSInteger)
index;
@optional
- (UIColor *)barChart:(BarView*)barChart colorForBarAtIndex: (NSInteger)
index; //获取柱形显示的颜色
- (NSString *)barChart:(BarView *)barChart textForBarAtIndex:(NSInteger)
index;
- (NSString *)barChart:(BarView *)barChart xLabelForBarAtIndex:(NSInteger)
index;
@end
//BarViewDelegate 协议
@protocol BarViewDelegate <NSObject>
@optional
- (void)animationDidEndForBarChart:(BarView *)barChart;
@end
@interface BarView : UIView{
    __weak id <BarViewDataSource> _dataSource;
    __weak id <BarViewDelegate> _delegate;
    CGFloat _xLabelMaxHeight;
    NSInteger _topValue;
    .....
    UIView *_xLabelView;
    UIView *_barTextView;
}
//属性
@property (weak, nonatomic) id <BarViewDataSource> dataSource;
@property (weak, nonatomic) id <BarViewDelegate> delegate;
.....
```



```
@property (nonatomic, assign) NSInteger incrementValue;
- (void)reloadData;
@end
```

(7) 打开 BarView.m 文件, 编写代码, 实现柱形图的绘制。使用的方法如表 3-4 所示。

表 3-4 BarView.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
reloadData	加载数据
setupBorders	创建边框
drawBorders	绘制边框
setupBars	创建柱形
animateBarAtIndex:	柱形的动画
drawBar:	绘制柱形
setupGrid	创建网格
drawGrid	绘制网格
setupYAxisLabels	创建 y 轴的标签
setupXAxisLabels	创建 x 轴的标签
setupBarTexts	创建柱形中的文本
displayAxisLabels	显示标签

可以将这些方法划分为两个步骤: 创建柱状图以及显示(动画效果)。这里需要讲解几个重要的方法(其他方法请读者参考源代码)。创建柱状图需要使用 initWithFrame:、reloadData、setupBorders、setupBorders、setupBars、setupGrid、setupYAxisLabels、setupXAxisLabels、setupBarTexts 方法。其中, initWithFrame:方法实现对柱状图的初始化。程序代码如下:

```
- (id)initWithFrame:(CGRect)frame
{
    if (!(self = [super initWithFrame:frame]))
        return self;
    self.animationDuration= 1.0; //设置动画持续时间
    self.hasGrids= YES;
    self.incrementValue= 10;
    self.barWidth= 20.0; //设置柱形的宽度
    self.barAlpha= 1.0;
    //颜色设置
    self.chartBorderColor= [UIColor blackColor];
    self.gridColor= [UIColor grayColor];
    //标签设置
    self.hasYLabels= YES;
    self.yLabelFont= [UIFont fontWithName:@"Helvetica" size:12.0];
    self.yLabelColor= [UIColor blackColor]; //设置 y 轴上标签的颜色
    self.xLabelFont= [UIFont fontWithName:@"Helvetica" size:12.0];
    self.xLabelColor= [UIColor blackColor]; //设置 x 轴上标签的颜色
    self.xLabelType= SimpleBarChartXLabelTypeVerticle;
    self.barTextFont= [UIFont fontWithName:@"Helvetica" size:12.0];
    self.barTextColor= [UIColor whiteColor]; //设置柱形的文本颜色
    self.barTextType= SimpleBarChartBarTextTypeTop;
    //创建可变数组对象
    _barPathLayers= [[NSMutableArray alloc] init];
    _barHeights= [[NSMutableArray alloc] init];
}
```

```

_barLabels= [[NSMutableArray alloc] init];
_barTexts= [[NSMutableArray alloc] init];
//创建并设置图层对象
_gridLayer= [CALayer layer];
[self.layer addSublayer:_gridLayer];           //添加图层对象
_barLayer= [CALayer layer];
[self.layer addSublayer:_barLayer];           //添加图层对象
_borderLayer= [CALayer layer];
[self.layer addSublayer:_borderLayer];       //添加图层对象
//创建并设置空白视图对象
_yLabelView= [[UIView alloc] init];
_yLabelView.alpha= 0.0;                       //设置透明度
[self addSubview:_yLabelView];
_xLabelView= [[UIView alloc] init];
_xLabelView.alpha= 0.0;                       //设置透明度
[self addSubview:_xLabelView];
_barTextView= [[UIView alloc] init];
_barTextView.alpha= 0.0;                     //设置透明度
[self addSubview:_barTextView];
return self;
}

```

reloadData 方法实现对数据的加载，即实现对柱形图的创建。程序代码如下：

```

- (void)reloadData
{
    if (_dataSource)
    {
        _numberOfBars = [_dataSource numberOfBarsInBarChart:self];
        //移除对象
        [_barHeights removeAllObjects];
        [_barLabels removeAllObjects];
        [_barTexts removeAllObjects];
        //遍历
        for (NSInteger i = 0; i < _numberOfBars; i++)
        {
            [_barHeights addObject:[NSNumber numberWithFloat:[_dataSource
            barChart:
            Self valueForKeyPath:i]]];           //添加对象
            if (_dataSource && [_dataSource respondsToSelector:
            @selector(barChart:xLabelForBarAtIndex:)]])
                [_barLabels addObject:[_dataSource barChart:self xLabelFor
                BarAtIndex:i]];           //添加对象
            if (_dataSource && [_dataSource respondsToSelector:
            @selector(barChart:textForBarAtIndex:)]])
                [_barTexts addObject:[_dataSource barChart:self textFor
                BarAtIndex:i]];           //添加对象
        }
        _maxHeight= [_barHeights valueForKeyPath:@"@max.self"];
        _minHeight= [_barHeights valueForKeyPath:@"@min.self"];
        _topValue= (self.incrementValue - (_maxHeight.integerValue % self.
        incrementValue)) + _maxHeight.integerValue;
        //判断 x 轴标签的类型
        switch (self.xLabelType)
        {
            case SimpleBarChartXLabelTypeVerticle:
            default:
                _xLabelRotation = 90.0;           //设置旋转度数
                break;
        }
    }
}

```

```

        case SimpleBarChartXLabelTypeHorizontal:
            _xLabelRotation = 0.0; //设置旋转度数
            break;
        case SimpleBarChartXLabelTypeAngled:
            _xLabelRotation = 45.0; //设置旋转度数
            break;
    }
    //遍历
    for (NSString *label in _barLabels)
    {
        NSDictionary *dic=[[NSDictionary alloc] initWithObjectsAndKeys:
            self.xLabelFont ,NSFontAttributeName, nil]; //创建字典对象
        CGSize labelSize=[label sizeWithAttributes:dic];
        //获取标签的尺寸大小
        CGFloat labelHeightWithAngle = sin(DEGREES_TO_RADIANS (_xLabel
            Rotation)) *labelSize.width;
        //判断 labelSize 的 height 属性值是否大于 labelHeightWithAngle
        if (labelSize.height > labelHeightWithAngle)
        {
            _xLabelMaxHeight = (_xLabelMaxHeight > labelSize.height) ?
                _xLabelMaxHeight : labelSize.height; //获取标签的最大高度
        }
        else{
            _xLabelMaxHeight = (_xLabelMaxHeight > labelHeightWith
                Angle)? _xLabelMaxHeight : labelHeightWithAngle;
            //获取标签的最大高度
        }
    }
    NSDictionary*dic=[[NSDictionary alloc] initWithObjectsAndKeys:self.
        yLabelFont ,NSFontAttributeName, nil];
    CGSize yLabelSize=self.hasYLabels ?[[NSString stringWithFormat:
        @"%i", _topValue]sizeWithAttributes:
        dic]: CGSizeZero;
    //设置 x、y 轴视图的框架
    _yLabelView.frame= CGRectMake(0.0,0.0,self.hasYLabels ? yLabel
        Size.width + 5.0 : 0.0,self.bounds.size.height);
    _xLabelView.frame= CGRectMake(_yLabelView.frame.origin.x + _y
        LabelView.frame.size.width,self.bounds.size.height - _x
        LabelMaxHeight,self.bounds.size.width - (_yLabelView. frame.
        origin.x + _yLabelView.frame.size.width),_xLabelMaxHeight);
    //设置图层的框架
    _gridLayer.frame= CGRectMake(_yLabelView.frame.origin.x +_yLabel
        View. frame. size.width,0.0,self.bounds.size.width - (_yLabelView.
        frame. origin.x + _yLabelView. frame.size. width), self. bounds.
        size.height - (_xLabelMaxHeight > 0.0 ? (_xLabelMaxHeight + 5.0) :
        0.0));
    _barLayer.frame= _gridLayer.frame;
    _borderLayer.frame= _gridLayer.frame;
    _barTextView.frame= _gridLayer.frame;
    [self setupBorders]; //创建边框
    [self drawBorders]; //绘制边框
    @autoreleasepool {
        [self setupBars];
        [self animateBarAtIndex:0];
    }
    if (self.hasGrids)
    {
        [self setupGrid]; //创建网格
    }

```

```

        [self drawGrid]; //绘制网格
    }
    [self setupYAxisLabels]; //创建 y 轴的标签
    [self setupXAxisLabels];
    [self setupBarTexts]; //创建柱形中的文本
}

```

setupBorders 方法实现对柱形图边框的设置，程序代码如下：

```

- (void)setupBorders
{
    if (_borderPathLayer != nil)
    {
        [_borderPathLayer removeFromSuperlayer]; //移除指定的图层对象
        _borderPathLayer = nil;
    }
    //设置上下左右 4 个点
    CGPoint bottomLeft= CGPointMake(CGRectGetMinX(_borderLayer.bounds),
    CGRectGetMinY(_borderLayer.bounds));
    CGPoint bottomRight= CGPointMake(CGRectGetMaxX(_borderLayer.bounds),
    CGRectGetMinY(_borderLayer.bounds));
    CGPoint topLeft= CGPointMake(CGRectGetMinX(_borderLayer.bounds),
    CGRectGetMaxY(_borderLayer.bounds));
    CGPoint topRight= CGPointMake(CGRectGetMaxX(_borderLayer.bounds),
    CGRectGetMaxY(_borderLayer.bounds));
    //绘制路径
    UIBezierPath *path = [UIBezierPath bezierPath];
    [path moveToPoint:bottomRight]; //设置开始点
    [path addLineToPoint:topRight]; //设置结束点
    [path addLineToPoint:topLeft];
    [path addLineToPoint:bottomLeft];
    [path addLineToPoint:bottomRight];
    //根据绘制的路径创建形状
    _borderPathLayer= [CAShapeLayer layer];
    _borderPathLayer.frame= _borderLayer.bounds; //设置框架
    _borderPathLayer.bounds= _borderLayer.bounds; //设置边界
    _borderPathLayer.geometryFlipped= YES;
    _borderPathLayer.path= path.CGPath; //设置路径
    _borderPathLayer.strokeColor= self.chartBorderColor.CGColor;
    _borderPathLayer.fillColor= nil;
    _borderPathLayer.lineWidth= 1.0f; //设置线的宽度
    _borderPathLayer.lineJoin= kCALineJoinBevel;
    [_borderLayer addSublayer:_borderPathLayer];
}

```

setupBars 方法实现对柱状图中柱形的创建，程序代码如下：

```

- (void)setupBars
{
    for (CAShapeLayer *layer in _barPathLayers)
    {
        if (layer != nil)
        {
            [layer removeFromSuperlayer]; //移除指定的图层对象
        }
    }
    [_barPathLayers removeAllObjects]; //移除所有的对象
    CGFloat barHeightRatio= _barLayer.bounds.size.height / (CGFloat) _

```

```

topValue;
CGFloat xPos= _barLayer.bounds.size.width / (_numberOfBars + 1);
//循环, 添加图层对象和对象
for (NSInteger i = 0; i < _numberOfBars; i++)
{
    CGPoint bottom= CGPointMake(xPos, _barLayer.bounds.origin.y);
    CGPoint top = CGPointMake(xPos, ((NSNumber *)[_barHeights object
    AtIndex:i]).floatValue * barHeightRatio);
    xPos+= _barLayer.bounds.size.width / (_numberOfBars + 1);
    UIBezierPath *path= [UIBezierPath bezierPath];    //创建贝塞尔路径
    [path moveToPoint:bottom];                        //设置开始点
    [path addLineToPoint:top];                        //设置结束点
    UIColor *barColor= [UIColor darkGrayColor];
    if (_dataSource && [_dataSource respondsToSelector: @selector
    (barChart:colorForBarAtIndex:)])
        barColor = [_dataSource barChart:self colorForBarAtIndex:i];
    //根据路径创建形状, 即柱形
    CAShapeLayer *barPathLayer= [CAShapeLayer layer];
    barPathLayer.frame= _barLayer.bounds;            //设置框架
    barPathLayer.bounds= _barLayer.bounds;
    barPathLayer.geometryFlipped = YES;
    barPathLayer.path= path.CGPath;                  //设置路径
    barPathLayer.strokeColor= [barColor colorWithAlphaComponent:
    self.self.barAlpha].CGColor;
    barPathLayer.fillColor= nil;
    barPathLayer.lineWidth= self.barWidth;           //设置线宽
    barPathLayer.lineJoin= kCALineJoinBevel;
    barPathLayer.hidden= YES;
    barPathLayer.shadowOffset= self.barShadowOffset; //设置阴影偏移量
    barPathLayer.shadowColor= self.barShadowColor.CGColor;
    barPathLayer.shadowOpacity= self.barShadowAlpha; //设置阴影透明度
    barPathLayer.shadowRadius= self.barShadowRadius;
    [_barLayer addSublayer:barPathLayer];
    [_barPathLayers addObject:barPathLayer];
}
}
}

```

setupGrid 方法实现对网格的创建, 即柱状图中横线的创建。程序代码如下:

```

- (void)setupGrid
{
    //判断_gridPathLayer 是否不为空
    if (_gridPathLayer != nil)
    {
        [_gridPathLayer removeFromSuperlayer];    //移除指定的图层
        _gridPathLayer = nil;
    }
    CGFloat gridUnit= _gridLayer.bounds.size.height / _topValue;
    CGFloat gridSeperation= gridUnit * (CGFloat)self.incrementValue;
    CGFloat yPos= gridSeperation;
    UIBezierPath *path= [UIBezierPath bezierPath];    //创建贝塞尔路径
    //循环设置点
    while (yPos <= _gridLayer.bounds.size.height)
    {
        CGPoint left= CGPointMake(0.0, yPos);
        CGPoint right= CGPointMake(_gridLayer.bounds.size.width, yPos);
        yPos+= gridSeperation;
    }
}

```



```

        [path moveToPoint:left]; //设置开始点
        [path addLineToPoint:right]; //设置结束点
    }
    //创建横线
    _gridPathLayer= [CAShapeLayer layer];
    _gridPathLayer.frame= _gridLayer.bounds; //设置框架
    _gridPathLayer.bounds= _gridLayer.bounds;
    _gridPathLayer.geometryFlipped= YES;
    _gridPathLayer.path= path.CGPath; //设置路径
    _gridPathLayer.strokeColor= self.gridColor.CGColor;
    _gridPathLayer.fillColor= nil;
    _gridPathLayer.lineWidth= 1.0f; //设置线宽
    _gridPathLayer.lineJoin= kCALineJoinBevel;
    [_gridLayer addSublayer:_gridPathLayer];
}

```

setupYAxisLabels 方法实现对 y 轴标签的创建。程序代码如下：

```

- (void)setupYAxisLabels
{
    if (!self.hasYLabels)
        return;
    //判断_yLabelView 的 alpha 属性值是否大于 0.0
    if (_yLabelView.alpha > 0.0)
    {
        _yLabelView.alpha = 0.0; //设置透明度
        [[_yLabelView subviews] makeObjectsPerformSelector:@selector
            (removeFromSuperview)];
    }
    CGFloat gridUnit= _gridLayer.bounds.size.height / _topValue;
    CGFloat gridSeperation= gridUnit * (CGFloat)self.incrementValue;
    NSDictionary *dic=[[NSDictionary alloc] initWithObjectsAndKeys: self.y
        Label Font ,NSFontAttributeName, nil];
    CGSize yLabelSize=[[NSString stringWithFormat:@"%i",_top Value]
        sizeWithAttributes:dic];
    CGFloat yPos= 0.0;
    NSInteger maxVal= _topValue;
    //判断 yPos 是否大于 _gridLayer.bounds.size.height 的属性值
    while (yPos < _gridLayer.bounds.size.height)
    {
        CGRect yLabelFrame= CGRectMake(0.0,0.0,yLabelSize. width,yLabel
            Size.height);
        //创建标签对象
        UILabel *yLabel = [[UILabel alloc] initWithFrame:yLabelFrame];
        yLabel.font = self.yLabelFont; //设置字体
        yLabel.backgroundColor= [UIColor clearColor];
        yLabel.textColor= self.yLabelColor;
        yLabel.textAlignment= NSTextAlignmentRight; //设置对齐方式
        yLabel.center= CGPointMake(yLabel.center.x, yPos);
        yLabel.text= [NSString stringWithFormat:@"%i", maxVal]; //设置文本内容
        [_yLabelView addSubview:yLabel];
        maxVal-= self.incrementValue;
        yPos+= gridSeperation;
    }
}

```

}

setupXAxisLabels 方法实现对 x 轴标签的创建。程序代码如下：

```

- (void)setupXAxisLabels
{
    //判断_barLabels 的 count 属性值是否为 0
    if (_barLabels.count == 0)
        return;
    //判断_xLabelViews 的 alpha 属性值是否为 0.0
    if (_xLabelView.alpha > 0.0)
    {
        _xLabelView.alpha = 0.0; //设置透明度
        [_xLabelViews.subviews]makeObjectsPerformSelector:@selector (removeFromSuperview)];
    }
    CGFloat xPos= _barLayer.bounds.size.width / (_numberOfBars + 1);
    //循环, 添加视图对象
    for (NSInteger i = 0; i < _numberOfBars; i++)
    {
        NSString *xLabelText= [_barLabels objectAtIndex:i];
        NSDictionary *dic=[NSDictionary alloc] initWithObjectsAndKeys:
            self.xLabelFont ,NSFontAttributeName, nil]; //创建字典对象
        CGSize xLabelSize=[xLabelText sizeWithAttributes:dic];
        CGRect xLabelFrame= CGRectMake(0.0,0.0,xLabelSize.width,xLabelSize.height);
        //创建并设置标签对象
        UILabel *xLabel= [[UILabel alloc] initWithFrame:xLabelFrame];
        xLabel.font= self.xLabelFont; //设置字体
        xLabel.backgroundColor= [UIColor clearColor];
        xLabel.textColor= self.xLabelColor;
        xLabel.textAlignment= NSTextAlignmentRight; //设置对齐方式
        xLabel.text= xLabelText; //设置文本内容
        xLabel.transform= CGAffineTransformMakeRotation
            (DEGREES_TO_RADIANS(- xLabelRotation));
        //判断标签的类型, 从而进行中心点的设置
        switch (self.xLabelType)
        {
            case SimpleBarChartXLabelTypeVerticle:
            default:
                xLabel.center = CGPointMake(xPos, (xLabelSize.width / 2.0));
                //设置中心位置
                break;
            case SimpleBarChartXLabelTypeHorizontal:
                xLabel.center = CGPointMake(xPos, _xLabelMaxHeight / 2.0);
                //设置中心位置
                break;
            case SimpleBarChartXLabelTypeAngled:
            {
                CGFloat labelHeightWithAngle= sin(DEGREES_TO_RADIANS
                    (_xLabelRotation)) * xLabelSize.width;
                xLabel.center= CGPointMake(xPos - (labelHeightWithAngle /

```

```

        2.0), labelHeightWithAngle / 2.0); //设置中心位置
        break;
    }
}
[_xLabelView addSubview:xLabel]; //添加视图对象
xPos+= _barLayer.bounds.size.width / (_numberOfBars + 1);
}
}

```

setupBarTexts 方法实现对柱形中显示文本的创建。程序代码如下：

```

-(void) setupBarTexts
{
    //判断_barLabels 的 count 属性值是否为 0
    if (_barTexts.count == 0)
        return;
    //判断_xLabelViews 的 alpha 属性值是否为 0.0
    if (_barTextView.alpha > 0.0)
    {
        _barTextView.alpha = 0.0; //设置透明度
        [[_barTextView subviews] makeObjectsPerformSelector: @selector
        (removeFromSuperview)];
    }
    CGFloat xPos= _barLayer.bounds.size.width / (_numberOfBars + 1);
    //遍历，实现视图对象的添加
    for (NSInteger i = 0; i < _numberOfBars; i++)
    {
        NSString *barLabelText = [_barTexts objectAtIndex:i];
        NSDictionary *dic=[[NSDictionary alloc] initWithObjectsAndKeys:
        self.barTextFont ,NSFontAttributeName, nil]; //创建字典对象
        CGSize barTextSize=[barLabelText sizeWithAttributes:dic];
        CGRect barTextFrame= CGRectMake(0.0,0.0,barTextSize.width,bar
        TextSize.height);
        UILabel *barText= [[UILabel alloc] initWithFrame:barTextFrame];
        //实例化标签对象
        barText.font= self.barTextFont;
        barText.backgroundColor= [UIColor clearColor]; //设置背景颜色
        barText.textColor= self.barTextColor;
        barText.textAlignment = NSTextAlignmentCenter;
        barText.text= barLabelText; //设置文本内容
        CGFloat barHeight= (_barLayer.bounds.size.height / (CGFloat)_top
        Value) * ((NSNumber *)[_barHeights objectAtIndex:i]).floatValue;
        //判断标签的类型，从而进行中心点的设置
        switch (self.barTextType)
        {
            case SimpleBarChartBarTextTypeTop:
            default:
                barText.center=CGPointMake(xPos, _barLayer.bounds.size.
                height- (barHeight- (barTextSize.height/2.0))); //设置中心位置
                break;
            case SimpleBarChartBarTextTypeRoof:
                barText.center = CGPointMake(xPos, _barLayer.bounds.size.

```

```

        height=(barHeight+(barTextSize.height/2.0)); //设置中心位置
        break;
    case SimpleBarChartBarTextTypeMiddle:
    {
        CGFloat minBarHeight= (_barLayer.bounds.size.height /
            (CGFloat)_topValue) * _minHeight.floatValue;
        barText.center= CGPointMake(xPos, _barLayer.bounds.size.
            height - (minBarHeight / 2.0)); //设置中心位置
        break;
    }
}
[_barTextView addSubview:barText];
xPos += _barLayer.bounds.size.width / (_numberOfBars + 1);
}
}

```

在本实例中要显示柱状图，都是以动画的形式显示的。需要使用 `drawBorders`、`animateBarAtIndex:`、`drawBar:`、`drawGrid`、`displayAxisLabels` 方法实现。其中，`drawBorders` 方法实现以绘制的动画形式显示柱状图上的边框。程序代码如下：

```

- (void)drawBorders
{
    //判断动画持续时间是否为 0.0
    if (self.animationDuration == 0.0)
        return;
    [_borderPathLayer removeAllAnimations]; //移除所有的动画
    //动画
    CABasicAnimation *pathAnimation= [CABasicAnimation animationWith
        KeyPath:@"strokeEnd"];
    pathAnimation.duration= self.animationDuration; //创建动画持续时间
    pathAnimation.fromValue= [NSNumber numberWithFloat:0.0f]; //设置开始值
    pathAnimation.toValue= [NSNumber numberWithFloat:1.0f]; //设置结束值
    [_borderPathLayer addAnimation:pathAnimation forKey:@"strokeEnd"];
}

```

`animateBarAtIndex:`方法实现以动画的形式显示柱形，程序代码如下：

```

- (void)animateBarAtIndex: (NSInteger)index
{
    //判断 index 是否大于等于_barPathLayers.count
    if (index >= _barPathLayers.count)
    {
        [self displayAxisLabels]; //显示标签
        return;
    }
    __block NSInteger i= index + 1;
    __weak BarView *weakSelf = self;
    //动画
    [CATransaction begin];
    //设置动画持续时间
    [CATransaction setAnimationDuration:(self.animationDuration / (CG Fl

```

```

    oat)_barPathLayers.count]);
    [CATransaction setAnimationTimingFunction:[CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseOut]];
    //设置完成动画时执行的块
    [CATransaction setCompletionBlock:^(
        [weakSelf animateBarAtIndex:i];
    )];
    CAShapeLayer *barPathLayer= [_barPathLayers objectAtIndex:index];
    barPathLayer.hidden= NO;                //不隐藏 barPathLayer 对象
    [self drawBar:barPathLayer];
    [CATransaction commit];
}

```

displayAxisLabels 方法实现对 x、y 轴以及柱形上标签的显示。

```

- (void)displayAxisLabels
{
    if (self.hasYLabels || _barTexts.count > 0 || _barLabels.count > 0)
    {
        //判断动画持续时间是否为 0.0
        if (self.animationDuration > 0.0)
        {
            __weak BarView *weakSelf = self;
            //实现动画
            [UIView animateWithDuration:self.animationDuration / 2.0
            delay:0.0 options:UIViewAnimationOptionCurveEaseOut animations:^(
            //设置透明度
                _yLabelView.alpha= 1.0;
                _xLabelView.alpha= 1.0;
                _barTextView.alpha= 1.0;
            ) completion:^(BOOL finished) {
                if (weakSelf.delegate && [weakSelf.delegate respondsToSelector:
                Selector:@selector(animationDidEndForBarChart:)])
                //调用 animationDidEndForBarChart:方法
                    [weakSelf.delegate animationDidEndForBarChart:weakSelf];
            }];
        }
        else
        {
            //设置透明度
            _yLabelView.alpha= 1.0;
            _xLabelView.alpha= 1.0;
            _barTextView.alpha= 1.0;
            if (_delegate && [_delegate respondsToSelector:@selector
            (animationDidEndForBarChart:)])
                [_delegate animationDidEndForBarChart:self];
            //调用 animationDidEndForBarChart:方法
        }
    }
    else
    {

```



```

    if (_delegate && [_delegate respondsToSelector: @selector(
        animationDidEndForBarChart:)]) {
        [_delegate animationDidEndForBarChart:self];
        //调用 animationDidEndForBarChart:方法
    }
}

```

(8) 打开 ViewController.h 文件, 编写代码, 实现头文件、遵守协议、对象、实例变量的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "BarView.h"
#import "NSMutableArray+MutableArray.h"
@interface ViewController : UIViewController<BarViewDataSource, BarView
Delegate>{
    NSArray *_values;
    BarView *_chart;
    NSArray *_barColors;
    NSInteger _currentBarColor;
}
@end

```

(9) 打开 ViewController.m 文件, 编写代码, 实例柱形图在界面的显示, 以及单击按钮后柱状图的重绘功能。使用的方法如表 3-5 所示。

表 3-5 ViewController.m文件中方法总结

方 法	功 能
loadView	加载视图
changeClicked	单击按钮
viewDidAppear:	视图即将加载时调用, 实现加载柱形图
numberOfBarsInBarChart:	获取柱形的个数
barChart:valueForBarAtIndex:	获取柱形的值, 用来控制器柱形的高度
barChart:textForBarAtIndex:	获取显示在柱形上的文本内容
barChart:xLabelForBarAtIndex:	获取显示在 x 轴上标签的内容
barChart:colorForBarAtIndex:	获取柱形显示的颜色

其中, loadView 方法实现视图的加载, 即实现图表的显示。程序代码如下:

```

- (void)loadView
{
    [super loadView];
    _values= @[30, 45, 44, 60, 95, 2, 8, 9];           //创建数组
    _barColors= @[[UIColor blueColor], [UIColor redColor], [UIColor
        blackColor], [UIColor orangeColor], [UIColor purpleColor], [UIColor
        greenColor]];                                   //创建数组
    _currentBarColor= 0;
    CGRect chartFrame= CGRectMake(0.0,0.0,300.0,300.0);
    //绘制有柱形图视图对象的创建并设置
    _chart= [[BarView alloc] initWithFrame:chartFrame];
    _chart.center= CGPointMake(self.view.frame.size.width / 2.0, self.
        view.frame.size.height / 2.0);
}

```

```

    _chart.delegate= self; //设置委托
    _chart.dataSource= self;
    _chart.barShadowOffset= CGSizeMake(2.0, 1.0);
    _chart.animationDuration= 1.0; //设置动画持续时间
    _chart.barShadowColor= [UIColor grayColor];
    _chart.barShadowAlpha= 0.5;
    _chart.barShadowRadius= 1.0;
    _chart.barWidth= 18.0; //设置柱形的宽度
    _chart.xLabelType= SimpleBarChartXLabelTypeVerticle;
    _chart.incrementValue= 10;
    _chart.barTextType= SimpleBarChartBarTextTypeTop;
    _chart.barTextColor= [UIColor whiteColor]; //设置柱形的文本颜色
    _chart.gridColor= [UIColor grayColor];
    [self.view addSubview:_chart];
    //创建并设置按钮对象
    UIButton *changeButton= [UIButton buttonWithType:UIButton TypeRounded Rect];
    changeButton.frame= CGRectMake(0.0, _chart.frame.origin.y + _chart.
    frame. size. height + 20.0, 100.0, 44.0);
    changeButton.center= CGPointMake(self.view.frame.size.width / 2.0,
    changeButton.center.y);
    [changeButton setTitle:@"Change" forState:UIControlStateNormal]; //设置标题
    [changeButton addTarget:self action:@selector(changeClicked) for
   ControlEvents:UIControlEventsTouchDown]; //添加动作
    [self.view addSubview:changeButton];
}

```

changeClicked 方法实现单击按钮后，重回柱形图的功能。程序代码如下：

```

- (void)changeClicked
{
    NSMutableArray *valuesCopy = _values.mutableCopy;
    [valuesCopy shuffle];
    _values = valuesCopy;
    //判断 _chart.xLabelType 是否为 SimpleBarChartXLabelTypeVerticle
    if (_chart.xLabelType == SimpleBarChartXLabelTypeVerticle)
        _chart.xLabelType = SimpleBarChartXLabelTypeHorizontal;
    else
        _chart.xLabelType = SimpleBarChartXLabelTypeVerticle;
    //设置柱状图的类型
    _currentBarColor = ++_currentBarColor % _barColors.count;
    [_chart reloadData];
}

```

【代码解析】

由于本实例中的代码和方法非常多，为了方便读者的阅读，笔者绘制了一个执行流程图，如图 3.4 所示。其中，单击运行按钮到柱状图以动画的形式显示在界面上，需要使用 initWithFrame:、reloadData、setupBorders、drawBorders、setupBars、animateBarAtIndex:、drawBar:、setupGrid、drawGrid、setupYAxisLabels、setupXAxisLabels、setupBarTexts、displayAxisLabels、loadView、viewDidAppear:、numberOfBarsInBarChart:、barChart:valueForBarAtIndex:、barChart:textForBarAtIndex:、barChart:xLabelForBarAtIndex:、barChart:colorForBarAtIndex:方法共同实现。

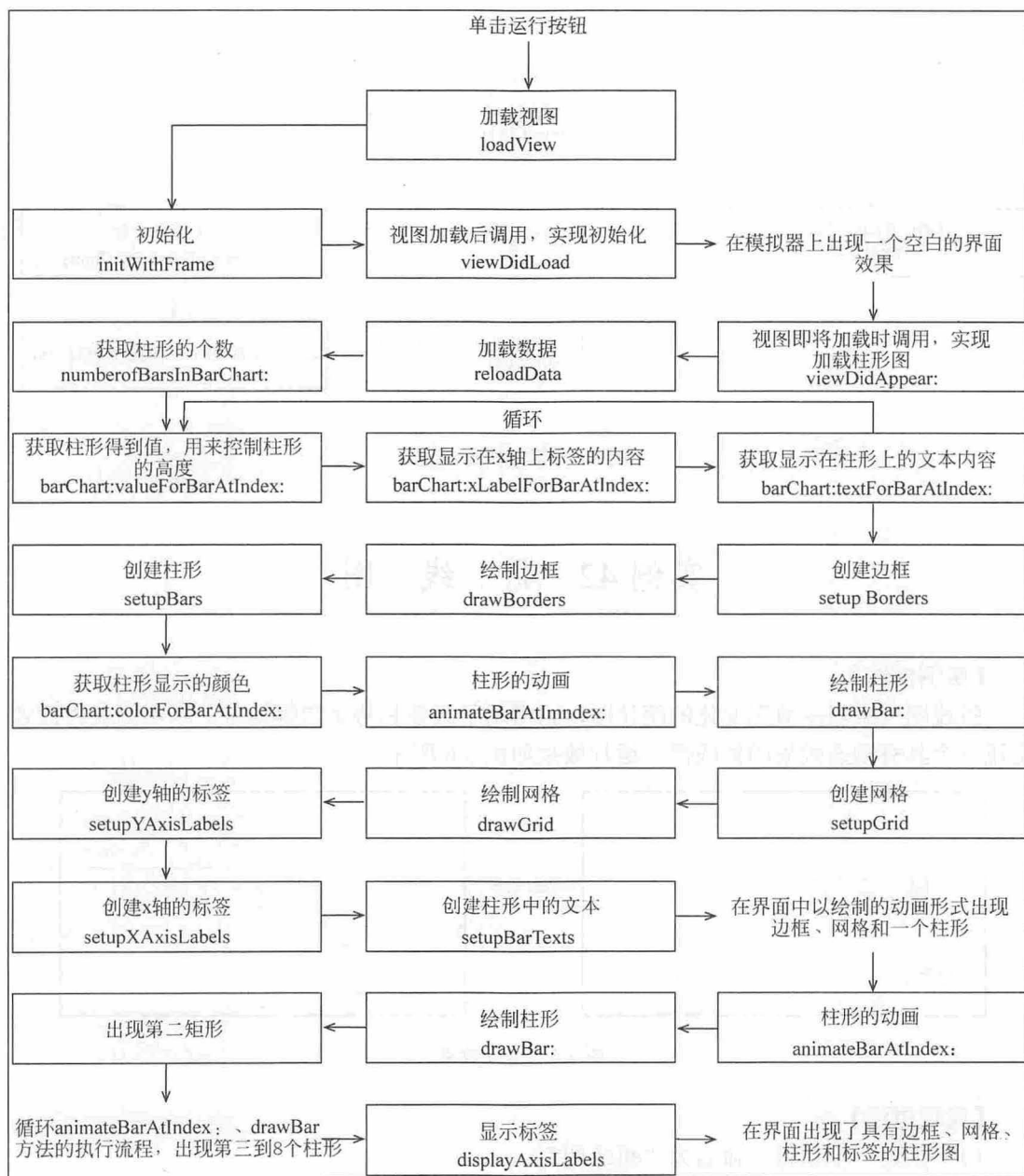


图 3.4 程序执行流程

这时会在界面显示一个柱状图和一个 Change 按钮。当用户单击此按钮后，会重新对柱状图进行绘制，并且颜色、柱形的高度以及内容都会发生改变，实现这些需要使用 shuffle、reloadData、setupBorders、drawBorders、setupBars、animateBarAtIndex:、drawBar:、setupGrid、drawGrid、setupYAxisLabels、setupXAxisLabels、setupBarTexts、displayAxisLabels、change Clicked、numberOfBarsInBarChart:、barChart:valueForBarAtIndex:、barChart:textForBarAtIndex:、barChart:xLabelForBarAtIndex:、barChart:colorForBarAtIndex: 方法共同实现。它们的程序执行流程如图 3.5 所示。

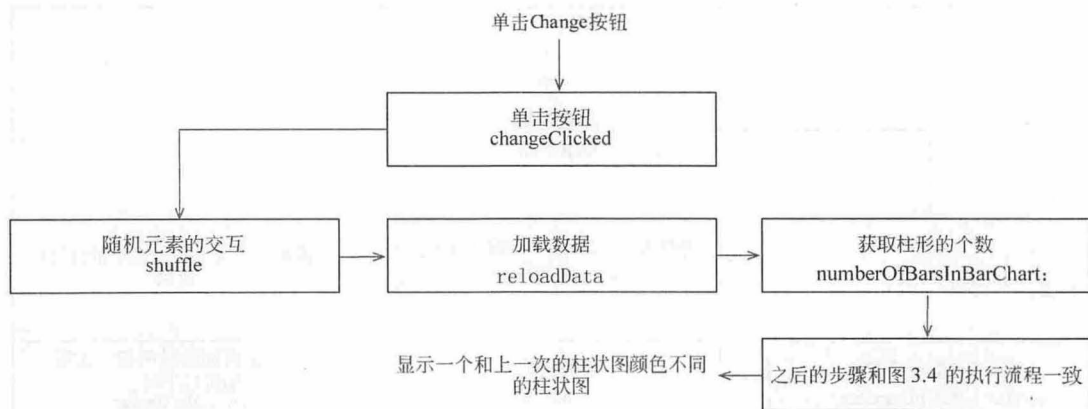


图 3.5 程序执行流程

实例 42 折线图

【实例描述】

折线图用来表示数据变化的统计图，它常用于股票趋势走向等地方。本实例就为读者实现一个具有动态效果的折线图。运行效果如图 3.6 所示。

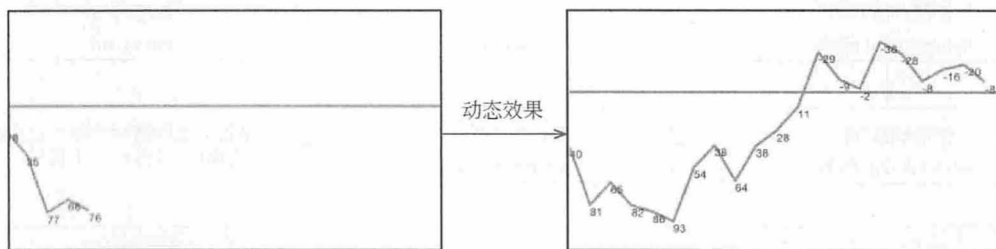


图 3.6 运行效果

【实现过程】

- (1) 创建一个项目，命名为“折线图”。
- (2) 创建一个基于 UIView 类的 View 类。
- (3) 打开 View.h 文件，编写代码，实现宏定义、属性的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
//宏定义
#define GraphColor [[UIColor blueColor] colorWithAlphaComponent:0.5]
#define str(index) [NSString stringWithFormat : @"%.f", [[self.values
objectAtIndex:(index)] floatValue] * kYScale]
#define point(x, y) CGPointMake((x) * kXScale, yOffset + (y) * kYScale)
@interface View : UIView
//属性
@property (nonatomic, readonly, strong) NSMutableArray *values;
@property (nonatomic, strong) dispatch_source_t timer;
  
```

@end

(4) 打开 View.m 文件，编写代码，实现具有动态效果的折线图。使用的方法如表 3-6 所示。

表 3-6 View.m文件中方法总结

方 法	功 能
CGAffineTransformMakeScaleTranslate	获取偏移量
awakeFromNib	在初始化后调用
updateValues	值的更新
drawRect:	绘制
drawAtPoint:withStr:	绘制文本

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，awakeFromNib 方法在初始化后调用，实现对定时器的创建以及设置。程序代码如下：

```
- (void)awakeFromNib
{
    [self setContentMode:UIViewContentModeRight];
    _values = [NSMutableArray array];
    __weak id weakSelf = self;
    double delayInSeconds = 0.25;
    //创建定时器
    self.timer =dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0,
    dispatch_get_main_queue());
    dispatch_source_set_timer(_timer,dispatch_walltime(NULL,0), (unsigned)
    (delayInSeconds * NSEC_PER_SEC), 0);
    //使用 dispatch_source_set_timer 函数设置定时器的参数
    //设置回调
    dispatch_source_set_event_handler(_timer, ^{[weakSelf updateValues];
    });
    dispatch_resume(_timer);
}
```

updateValues 方法实现在每隔 0.25 秒后对数值的更新。程序代码如下：

```
- (void)updateValues
{
    double nextValue = sin(CFAbsoluteTimeGetCurrent()) + ((double)rand() /
    (double)RAND_MAX);
    [self.values addObject: [NSNumber numberWithInt:nextValue]]; //添加对象
    CGSize size = self.bounds.size; //获取边界的尺寸
    CGFloat maxDimension = size.width;
    NSUInteger maxValues = (NSUInteger)floorl(maxDimension / kXScale);
    if ([self.values count] > maxValues) {
        [self.values removeObjectsWithRange:
        NSMakeRange(0, [self.values count] - maxValues)]; //移除对象
    }
    [self setNeedsDisplay];
}
```

drawRect:方法实现折线图的绘制。程序代码如下：

```
- (void)drawRect:(CGRect) rect
{
    if ([self.values count] == 0) {
        return;
    }
}
```



```

    }
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    CGContextSetStrokeColorWithColor(ctx, [GraphColor CGColor]);
    //设置线的颜色
    CGContextSetLineJoin(ctx, kCGLineJoinRound);
    CGContextSetLineWidth(ctx, 2); //设置线宽
    CGMutablePathRef path = CGPathCreateMutable();
    CGFloat yOffset = self.bounds.size.height / 2;
    CGAffineTransform transform = CGAffineTransformMakeScaleTranslate(
        kXScale, kYScale, 0, yOffset);
    //中间轴的绘制
    CGPathMoveToPoint(path, &transform, 0, 0); //设置开始点
    CGPathAddLineToPoint(path, &transform, self.bounds.size.width, 0);
    //设置结束点
    CGFloat y = [[self.values objectAtIndex:0] floatValue];
    CGPathMoveToPoint(path, &transform, 0, y);
    [self drawAtPoint:point(0, y) withStr:str(0)]; //绘制
    for (NSUInteger x = 1; x < [self.values count]; ++x) {
        y = [[self.values objectAtIndex:x] floatValue];
        CGPathAddLineToPoint(path, &transform, x, y); //设置结束点
        [self drawAtPoint:point(x, y) withStr:str(x)];
    }
    CGContextAddPath(ctx, path); //添加路径
    CGPathRelease(path);
    CGContextStrokePath(ctx);
}

```

drawAtPoint:str 方法实现文本的绘制。程序代码如下：

```

- (void)drawAtPoint:(CGPoint)point withStr:(NSString *)str
{
    [self drawAtPoint:point withAttributes:@{NSFontAttributeName:[UIFont
    sys temFontOfSize:8], NSStrokeColorAttributeName:GraphColor}];
    //绘制
}

```

【代码解析】

本实例关键功能是动态折线图的绘制以及指定点文本的绘制。下面依次讲解这两个知识点。

1. 动态折线图的绘制

在本实例中折线的绘制是使用多个线段组成的。在每隔 0.25 秒后就会执行一次以下的程序。从而实现动态折线的绘制。

```

CGPathMoveToPoint(path, &transform, 0, y);
[self drawAtPoint:point(0, y) withStr:str(0)];
for (NSUInteger x = 1; x < [self.values count]; ++x) {
    y = [[self.values objectAtIndex:x] floatValue];
    CGPathAddLineToPoint(path, &transform, x, y);
    [self drawAtPoint:point(x, y) withStr:str(x)];
}

```

2. 指定点文本的绘制

通过指定的某一点绘制文本，需要使用 **drawAtPoint:withAttributes:** 方法。其语法形式如下：

```
- (void)drawAtPoint:(CGPoint)point withAttributes:(NSDictionary *)attrs;
```

其中, (CGPoint)point 表示指定的点; (NSDictionary *)attrs 表示文本属性。在本实例中就是使用了 drawAtPoint:withAttributes:方法实现了在固定的点对文本进行显示。代码如下:

```
[str drawAtPoint:pointwithAttributes:@{NSFontAttributeName:[UIFont systemFontOfSize:8],
NSStrikeColorAttributeName:GraphColor}];
```

其中, point 表示指定的点。@{NSFontAttributeName:[UIFont systemFontOfSize:8], NSStrikeColorAttributeName:GraphColor}表示文本的属性。

实例 43 波形图

【实例描述】

本实例主要实现一个具有动画的波形图。当用户点击“暂停”按钮, 波形图的动画就会停止。当用户点击“开始”按钮, 停止的波形图就会运动。运行效果如图 3.7 所示。

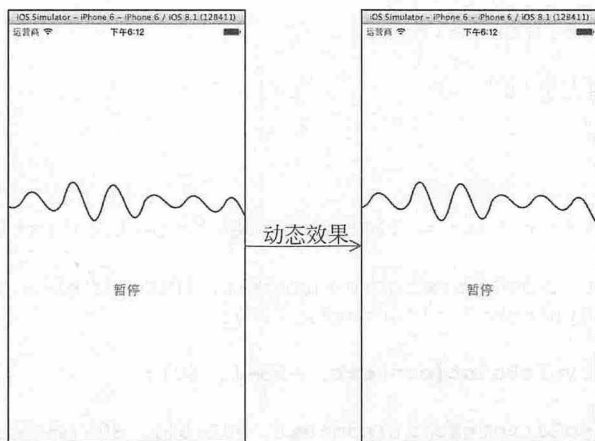


图 3.7 运行效果

【实现过程】

- (1) 创建一个项目, 命名为“波形图”。
- (2) 创建一个基于 UIView 类的 Wave 类。
- (3) 打开 Wave.h 文件, 编写代码, 实现实例变量以及方法的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface WaveView : UIView{
    float swing;
    int i;
    float j;
}
- (void)callDraw:(float)f;
@end
```

- (4) 打开 Wave.m 文件, 编写代码, 实现动态效果的波形图。使用的方法如表 3-7 所示。

表 3-7 Wave.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
callDraw:	判断波的幅度，并调用绘制方法
drawRect:	绘制波

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，callDraw:方法实现对波的幅度的判断。程序代码如下：

```
- (void)callDraw:(float)f
{
    //判断 f 是否小于 1
    if (f < 1) {
        swing = f;
    }else {
        swing = 1;
    }
    [self setNeedsDisplay];           //重新绘制
}
```

drawRect:方法实现对波的绘制。程序代码如下：

```
- (void)drawRect:(CGRect)rect
{
    //判断 i 是否大于 55
    if (i > 55) {
        i = 0;
        j = 0;
    }
    CGContextRef context = UIGraphicsGetCurrentContext();
    //设置
    CGContextSetStrokeColorWithColor(context, [UIColor blackColor].CGColor);
    CGContextSetLineWidth(context, 2.0);
    //绘制波
    CGContextMoveToPoint(context, -55+i, 40);
    //添加弧线
    CGContextAddCurveToPoint(context, -32.5+i, 30-(swing*20), -22.5+i, 50+(swing*20), 0+i, 40);
    CGContextMoveToPoint(context, 0+i, 40);
    //添加弧线
    CGContextAddCurveToPoint(context, 22.5+i, 30-(swing*(20+j)), 32.5+i, 50+(swing*(20+j)), 55+i, 40);
    CGContextMoveToPoint(context, 55+i, 40);
    //添加弧线
    CGContextAddCurveToPoint(context, 77.5+i, 30-(swing*80), 87.5+i, 50+(swing*80), 110+i, 40);
    CGContextMoveToPoint(context, 110+i, 40);           //设置开始点
    CGContextAddCurveToPoint(context, 132.5+i, 30-(swing*(80-j)), 142.5+i, 50+(swing*(80-j)), 165+i, 40);
    CGContextMoveToPoint(context, 165+i, 40);
    CGContextAddCurveToPoint(context, 187.5+i, 30-(swing*20), 197.5+i, 50+(swing*20), 220+i, 40);
    CGContextMoveToPoint(context, 220+i, 40);           //设置开始点
    CGContextAddCurveToPoint(context, 242.5+i, 30-(swing*20), 252.5+i, 50+
```

```

(swing* (20+j)), 275+i, 40);
CGContextMoveToPoint(context, 275+i, 40);
CGContextAddCurveToPoint(context, 297.5+i, 30-(swing*20), 307.5+i, 50+
(swing* 80), 330+i, 40);
CGContextStrokePath(context); //绘制
i++;
j = j+0.727;
}

```

(5) 打开 `ViewController.h` 文件, 编写代码, 实现头文件、对象以及实例变量的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "WaveView.h"
@interface ViewController : UIViewController{
    WaveView *view;
    NSTimer *timer; //声明时间定时器
    UIButton *open;
    BOOL on_off;
}
@end

```

(6) 打开 `ViewController.m` 文件, 编写代码, 实现动态波的效果。使用的方法如表 3-8 所示。

表 3-8 ViewController.m文件中方法总结

方 法	功 能
<code>viewDidLoad</code>	视图加载后调用, 实现初始化
<code>backButton</code>	单击按钮, 控制动态波
<code>detectionVoice</code>	为波的幅度赋值

其中, `viewDidLoad` 方法实现对绘制有波形图的视图的对象、按钮等进行创建和设置。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    view = [[WaveView alloc] initWithFrame:CGRectMake(0, 200, 320, 80)];
    view.backgroundColor = [UIColor clearColor]; //设置背景颜色
    view.clipsToBounds = YES;
    [self.view addSubview:view];
    //设置定时检测
    timer = [NSTimer scheduledTimerWithTimeInterval:0.01 target:self
    selector:@selector(detectionVoice) userInfo:nil repeats:YES];
    //创建定时器

    //创建并设置按钮对象
    open = [UIButton buttonWithType:UIButtonTypeCustom];
    open.frame = CGRectMake(140, 350, 40, 20); //设置框架
    [open setTitleColor:[UIColor blueColor] forState:UIControlStateNormal];
    [open setTitle:@"暂停" forState:UIControlStateNormal]; //设置标题
    [open addTarget:self action:@selector(backButton) forControlEvents:
    UIControlEventTouchUpInside];
    [self.view addSubview:open];
}

```

backButton 方法实现对波形的动画效果的控制。程序代码如下：

```
- (void)backButton
{
    if (on_off) {
        [open setTitle:@"暂停" forState:UIControlStateNormal];
        //开启定时器
        [timer setFireDate:[NSDate distantPast]];
    }else {
        [open setTitle:@"开始" forState:UIControlStateNormal];
        //关闭定时器
        [timer setFireDate:[NSDate distantFuture]];
    }
    on_off = !on_off;
}
```

【代码解析】

本实例关键功能是波的绘制以及波的动力效果。下面着重讲解波的绘制。

波形其实就是一条曲线。曲线的绘制需要使用 CGContext 的 CGContextAddCurveToPoint 函数，其语法形式如下：

```
void CGContextAddCurveToPoint (
    CGContextRef c,
    CGFloat cplx,
    CGFloat cply,
    CGFloat cp2x,
    CGFloat cp2y,
    CGFloat x,
    CGFloat y
);
```

其中，参数说明如下：

- ❑ CGContextRef c 表示上下文；
- ❑ CGFloat cplx 表示第一个指定点的 x 坐标；
- ❑ CGFloat cply 表示第一个指定点的 y 坐标；
- ❑ CGFloat cp2x 表示第二个指定点的 x 坐标；
- ❑ CGFloat cp2y 表示第二个指定点的 y 坐标；
- ❑ CGFloat x 表示终点 x 的坐标；
- ❑ CGFloat y 表示终点 y 的坐标。

注意在使用 CGContextAddCurveToPoint 函数之前，必须要先使用 CGContextMoveToPoint 函数指定起始点。在此代码中就是使用 CGContextAddCurveToPoint 函数实现了波形的绘制。代码如下：

```
CGContextAddCurveToPoint(
    context,
    -32.5+i,
    30-(swing*20),
    -22.5+i,
    50+(swing*20),
    0+i,
```


40

);

其中，参数说明如下：

- context 表示上下文；
- $-32.5+i$ 表示第一个指定点的 x 坐标；
- $30-(swing*20)$ 表示第一个指定点的 y 坐标；
- $-22.5+i$ 表示第二个指定点的 x 坐标；
- $50+(swing*20)$ 表示第二个指定点的 y 坐标；
- $0+i$ 表示终点 x 的坐标；
- 40 表示终点 y 的坐标。

实例 44 油 量 表

【实例描述】

在汽车中，油量表是很常见的，它用来表示油量的多少。在本实例中就为读者实现一个类似于汽车中的油量表。当用户单击 Change 按钮后，油量表中的数字以及指针（线）就会发生改变。运行效果如图 3.8 所示。

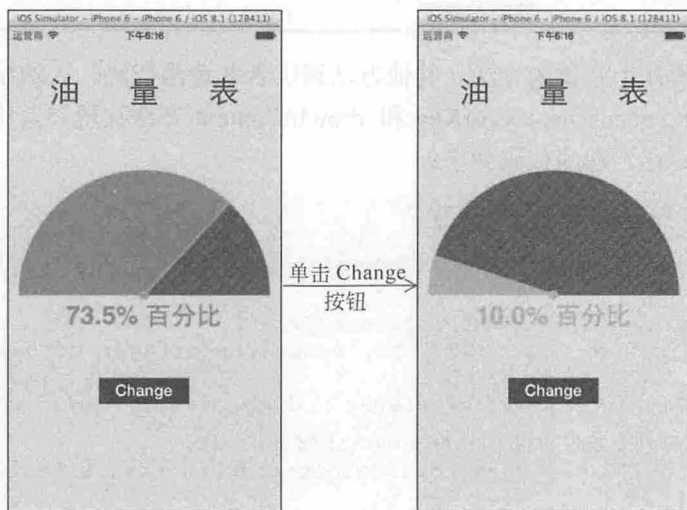


图 3.8 运行效果

【实现过程】

- (1) 创建一个项目，命名为“油量表”。
- (2) 添加 CoreText.framework 框架到创建的项目中。
- (3) 创建一个基于 CALayer 类的 SemicircularLayer 类。
- (4) 打开 SemicircularLayer.h 文件，编写代码，实现头文件、宏定义、属性以及方法的声明。程序代码如下：

```
#import <QuartzCore/QuartzCore.h>
```

```

#import <CoreText/CoreText.h> //头文件
//宏定义
#define DEG2RAD(angle) angle*M_PI/180.0
#define INITIAL_ANGLE 180
#define ENDING_ANGLE 0
#define CENTER_WIDTH 8
@interface SemicircularLayer : CALayer
//属性
@property(nonatomic) CGFloat percentage;
@property(nonatomic, strong) NSString *text;
.....
@property(nonatomic) CGFloat fontSize;
- (id)initWithLayer:(id)aLayer; //初始化方法
@end

```

(5) 打开 **SemicircularLayer.m** 文件，编写代码，实现对油量表的绘制以及动画效果的实现。使用的方法如表 3-9 所示。

表 3-9 SemicircularLayer.m文件中方法总结

方 法	功 能
makeAnimationForKey:	获取动画
actionForKey:	获取行为对象
initWithLayer:	初始化图层
needsDisplayForKey:	修改了指定的键后判断是否需要重新显示该层
drawInContext:	绘制油量表

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。油量表的绘制需要使用 `initWithLayer:`、`needsDisplayForKey:`和 `drawInContext:`方法实现。其中，`initWithLayer:`表示对图层的初始化。程序代码如下：

```

- (id)initWithLayer:(id)aLayer
{
    if (self = [super initWithLayer:aLayer])
    {
        //判断 aLayer 是否在 SemicircularLayer 中
        if ([aLayer isKindOfClass:[SemicircularLayer class]])
        {
            SemicircularLayer *layer = (SemicircularLayer *)aLayer;
            //判断是否实现了 setContentsScale:方法
            if ([layer respondsToSelector:@selector(setContentsScale:)])
            {
                layer.contentsScale = [[UIScreen mainScreen] scale];
            }
            self.percentage = layer.percentage;
            self.text = layer.text; //设置文本内容
            self.mainColor = layer.mainColor;
            self.secondaryColor = layer.secondaryColor;
            self.lineColor = layer.lineColor; //设置线的颜色
            self.fontName = layer.fontName;
            self.fontSize = layer.fontSize; //设置字体的大小
        }
    }
    return self;
}

```

drawInContext:方法实现对油量表的绘制。程序代码如下

```

-(void)drawInContext:(CGContextRef)ctx
{
    NSDictionary *dic=[NSDictionary dictionaryWithObjectsAndKeys:[UIFont
    fontWithName:self.fontName size:self.fontSize],NSFontAttributeName ,
    nil];
    CGSize sizeControl=[@"sample" boundingRectWithSize:self.frame.size
    options:NSStringDrawingUsesLineFragmentOrigin | NSStringDrawingUses
    FontLeading attributes:dic context:nil].size;
    CGPoint center = CGPointMake( self.bounds.size.width/2, self.bounds.
    size.height - sizeControl.height - 10.0 );
    CGFloat radius = MIN( center.x, center.y ) - 1;
    CGFloat startingAngleRad = DEG2RAD( INITIAL_ANGLE );
    CGFloat endingAngleRad = DEG2RAD( ENDING_ANGLE );
    CGFloat currentAngle = INITIAL_ANGLE + ( INITIAL_ANGLE * self.
    percentage/100.0 );
    CGFloat currentAngleRad = DEG2RAD( currentAngle );
    CGPoint startingPoint = CGPointMake( center.x + radius * cosf
    (startingAngleRad), center.y + radius * sinf(startingAngleRad) );
    //开始点
    CGPoint endPoint = CGPointMake( center.x + radius * cosf
    (currentAngleRad) , center.y + radius * sinf(currentAngleRad) );
    //结束点

    //绘制半圆
    CGContextBeginPath( ctx );
    CGContextMoveToPoint( ctx, center.x, center.y );
    CGContextAddLineToPoint( ctx, startingPoint.x, startingPoint.y );
    CGContextAddArc( ctx, center.x, center.y, radius, startingAngleRad,
    currentAngleRad, NO );
    CGContextClosePath( ctx );
    CGContextSetFillColorWithColor( ctx, self.mainColor );
    CGContextSetStrokeColorWithColor( ctx, self.mainColor );
    CGContextSetLineWidth( ctx, 1 );
    CGContextDrawPath( ctx, kCGPathFillStroke );
    //绘制背景
    if( self.percentage < 100.0 )
    {
        CGContextBeginPath( ctx );
        CGContextMoveToPoint( ctx, center.x, center.y );
        CGContextAddLineToPoint( ctx, endPoint.x, endPoint.y );
        CGContextAddArc( ctx, center.x, center.y, radius, currentAngleRad,
        endingAngleRad, NO );
        CGContextClosePath( ctx );
        CGContextSetFillColorWithColor( ctx, self.secondaryColor.CGColor );
        CGContextSetStrokeColorWithColor( ctx, self.secondaryColor.CGColor );
        CGContextSetLineWidth( ctx, 1 );
        CGContextDrawPath( ctx, kCGPathFillStroke );
    }
    // 绘制中心点和进程线
    CGContextBeginPath( ctx );
    CGContextMoveToPoint( ctx, center.x, center.y );
    //设置开始点
    CGRect rect = CGRectMake( center.x - CENTER_WIDTH/2, center.y -
    CENTER_WIDTH/2, CENTER_WIDTH, CENTER_WIDTH );

```

```

CGContextAddEllipseInRect( ctx, rect ); //添加圆
//进程线
CGContextMoveToPoint( ctx, center.x, center.y );
CGContextAddLineToPoint( ctx, endPoint.x, endPoint.y );
CGContextClosePath( ctx ); //关闭路径
CGContextSetFillColorWithColor( ctx, self.lineColor );
CGContextSetStrokeColorWithColor( ctx, self.lineColor ); //设置线的颜色
CGContextSetLineCap( ctx, kCGLineCapRound );
CGContextSetLineWidth( ctx, 3 ); //设置线宽
CGContextDrawPath( ctx, kCGPathFillStroke );
//对文本的绘制
CGContextSetTextMatrix( ctx, CGAffineTransformMakeScale( 1.0, -1.0 ));
NSString *str = [NSString stringWithFormat:@"%0.1f%@ %@", self.
percentage, @"%", self.text];
NSDictionary *arr=[NSDictionary dictionaryWithObjectsAndKeys:[UIFont
fontName:self.fontName size:self.fontSize],NSFontAttributeName ,
nil]; //创建字典
//绘制尺寸
CGSize strSize=[str boundingRectWithSize:self.frame.size options:
NSStringDrawingUsesLineFragmentOrigin|NSStringDrawingUsesFontLeading
attributes:arr context:nil].size; //获取尺寸
CTFontRef sysUIFont = CTFontCreateWithName( (__bridge CFStringRef)
self.fontName, self.fontSize, NULL );
NSDictionary *attributesDict = [NSDictionary dictionaryWithObjects
AndKeys:(__bridge id)sysUIFont, (id)kCTFontAttributeName, self.main
Color, (id)kCTForegroundColorAttributeName, nil]; //创建字典
NSAttributedString *attributedString = [[NSAttributedString alloc] init
WithString:str attributes:attributesDict];
//创建一个文本行对象
CTLineRef lineref = CTLineCreateWithAttributedString( (__bridge
CFAttributedStringRef)attributedString );
CGContextSetTextPosition( ctx, center.x - ( strSize.width/2.0 ),
center.y + strSize.height ); //设置位置
CTLineDraw( lineref, ctx ); //绘制
CFRelease( lineref );
CFRelease( sysUIFont );
}

```

动画效果的实现需要使用 `makeAnimationForKey:` 和 `actionForKey:` 方法。其中，`makeAnimationForKey:` 方法实现获取动画对象。程序代码如下：

```

-(CABasicAnimation *)makeAnimationForKey:(NSString *)key
{
    CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:key];
    anim.fromValue = [[self presentationLayer] valueForKey:key]; //设置开始值
    anim.timingFunction = [CAMediaTimingFunction functionWithName:
kCAMediaTimingFunctionEaseOut];
    anim.duration = 0.5;
    return anim;
}

```

(6) 创建一个基于 `UIView` 类的 `Semicircular` 类。

(7) 打开 `Semicircular.h` 文件，编写代码，实现头文件、对象、属性以及方法的声明。程序代码如下：

```

#import <UIKit/UIKit.h>

```

```

#import "SemicircularLayer.h"                                //头文件
@interface Semicircular : UIView{
    CALayer *_mainLayer;
}
//属性
@property(n nonatomic) CGFloat percentage;
@property(n nonatomic, strong) NSString *text;
.....
@property(n nonatomic) CGFloat fontSize;
-(void) refresh;                                           //刷新方法
@end

```

(8) 打开 Semicircular.m 文件, 编写代码, 实现对一些属性的设置以及刷新功能。使用的方法如表 3-10 所示。

表 3-10 Semicircular.m文件中方法总结

方 法	功 能
setPercentage:	设置百分比
setText:	设置文本
setMainColor:	设置线滑动过的颜色
setSecondaryColor:	设置线没有滑动过的颜色
setLineColor:	设置线的颜色
setFontName:	设置字体的名称
setFontSize:	设置字体的尺寸
initialize	初始化
initWithFrame:	使用框架初始化
initWithCoder:	初始化、实例化对象
refresh	刷新

其中, setPercentage:方法执行在单击按钮进行刷新后对百分比的设置。程序代码如下:

```

-(void) setPercentage:(CGFloat)newValue
{
    if( newValue > 100.0 ) newValue = 100.0;
    else if( newValue < 0.0 ) newValue = 0.0;
    percentage = newValue;                                //设置百分比
    [self refresh];
}

```

initialize 方法实现一些初始化的设置。程序代码如下:

```

-(void) initialize
{
    _mainLayer = [SemicircularLayer layer];
    [self.layer addSublayer:_mainLayer];
    self.text = [NSString string];
    self.fontName = @"Helvetica";                        //设置字体
    self.fontSize = 30.0;                                //设置字体大小
}

```

refresh 方法实现单击按钮后的刷新效果。程序代码如下:

```

-(void) refresh
{
    _mainLayer.frame = self.bounds;
}

```



```

SemicircularLayer *layer = (SemicircularLayer *) _mainLayer;
layer.percentage = self.percentage;
layer.text = self.text;                                //设置文本内容
layer.mainColor = self.mainColor.CGColor;
layer.secondaryColor = self.secondaryColor;
layer.lineColor = self.lineColor.CGColor;              //设置线的颜色
layer.fontName = self.fontName;
layer.fontSize = self.fontSize;                        //设置字体大小
}

```

(9) 打开 ViewController.h 文件，编写代码，实现头文件、宏定义、插座变量、实例变量以及方法的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "Semicircular.h"                                //头文件
//宏定义
#define MAIN_ORANGE [UIColor colorWithRed:0.83 green:0.38 blue:0.0 alpha:1.0]
#define LINE_ORANGE [UIColor orangeColor]
#define MAIN_RED [UIColor colorWithRed:0.70 green:0.0 blue:0.0 alpha:1.0]
#define LINE_RED [UIColor redColor]
#define MAIN_GREEN [UIColor colorWithRed:0.47 green:0.7 blue:0.0 alpha:1.0]
#define LINE_GREEN [UIColor colorWithRed:0.0 green:0.7 blue:0.0 alpha:1.0]
@interface ViewController : UIViewController{
    //变量
    IBOutlet Semicircular *chart;
    CGFloat _percentage;
}
- (IBAction)Change: (id) sender;
@end

```

(10) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 3.9 所示。

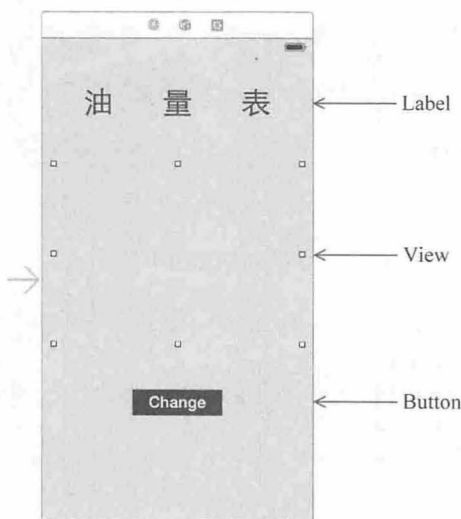


图 3.9 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 3-11 所示。

表 3-11 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 浅灰色	
Label	Text: 油 量 表 Font: System 36.0 Alignment: 居中	与插座变量 valueLabel 关联
View	Background: 透明色	Class: Semicircular 与插座变量 chart 关联
Button	Title: Change Font: System Bold 19.0 Text Color: 白色 Background: 黑色	与动作 Change:关联

(11) 打开 ViewController.m 文件, 编写代码, 实现对界面初始化的设置以及实现单击按钮后的响应。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    _percentage = 10.0;
    [chart setMainColor:MAIN_ORANGE];
    [chart setLineColor:LINE_ORANGE]; //设置线的颜色
    [chart setSecondaryColor:[UIColor darkGrayColor]];
    [chart setFontName:@"Helvetica-Bold"]; //设置字体
    [chart setFontSize:30.0];
    [chart setText:@"百分比"]; //设置文本内容
    [chart setPercentage:73.5];
}

//单击按钮
- (IBAction)Change:(id)sender {
    [chart setPercentage:_percentage];
    //判断 chart 的 percentage 是否大于 85
    if( [chart percentage] > 85 )
    {
        [chart setMainColor:MAIN_RED];
        [chart setLineColor:LINE_RED];
    } else if( [chart percentage] > 65 ) { //判断 chart 的 percentage 是否大于 65
        [chart setMainColor:MAIN_ORANGE];
        [chart setLineColor:LINE_ORANGE]; //设置线的颜色
    } else {
        [chart setMainColor:MAIN_GREEN];
        [chart setLineColor:LINE_GREEN]; //设置线的颜色
    }
    _percentage +=10;
    if( _percentage > 100.0 )
        _percentage -= 101.0;
}

```

【代码解析】

本实例关键功能是油量表颜色的改变以及油量表的动画效果。下面依次讲解这两个知识点。

1. 油量表颜色的改变

在本实例中油量表颜色的改变是通过对百分比值的判断实现的。代码如下：

```
if( [chart percentage] > 85 )
{
    [chart setMainColor:MAIN_RED];
    [chart setLineColor:LINE_RED];
} else if( [chart percentage] > 65 ) {
    [chart setMainColor:MAIN_ORANGE];
    [chart setLineColor:LINE_ORANGE];
} else {
    [chart setMainColor:MAIN_GREEN];
    [chart setLineColor:LINE_GREEN];
}
```

2. 油量表的动画效果

在本实例中看到的油量表的动画效果是通过使用 CABasicAnimation 动画效果实现的。代码如下：

```
CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:key];
anim.fromValue = [[self presentationLayer] valueForKey:key]; //设置开始值
anim.timingFunction = [CAMediaTimingFunction functionWithName: kCAMediaTimingFunctionEaseOut];
anim.duration = 0.5;
```

第4章 动画

动画的概念不同于一般意义上的动画片，动画是一种综合艺术，它是集合了绘画、文学等众多艺术门类于一身的艺术表现形式。在 iOS 中使用动画使枯燥的操作以及界面变得生动形象。本章将主要讲解关于动画的实例，例如吃豆豆、打砖块以及钟表等各种动画效果。

实例 45 飘落的雪花

【实例描述】

冬天，漫天飞舞的雪花非常漂亮，本实例就为读者实现飘落的雪花效果。运行效果如图 4.1 所示。



图 4.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“飘落的雪花”。
- (2) 添加图像 1.jpg、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现对象以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    UIImage* flakeImage;
}
- (void)onTimer;
```

```
@end
```

(4) 打开 Main.storyboard 文件，从视图库中拖动 Image View 图像视图到设计界面上。将 Image 属性设置为 1.jpg。

(5) 打开 ViewController.m 文件，编写代码，实现雪花的飘落效果。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    flakeImage = [UIImage imageNamed:@"2.png"];
    //创建定时器
    [NSTimer scheduledTimerWithTimeInterval:(0.05) target:self selector:
    @selector(onTimer) userInfo:nil repeats:YES];
}

- (void)onTimer
{
    //创建图像视图对象
    UIImageView* flakeView = [[UIImageView alloc] initWithImage: flakeImage];
    //定义图像的的开始和结束坐标 x 值
    int startX = round(random() % 320);
    int endX = round(random() % 320);
    double scale = 1 / round(random() % 100) + 1.0;
    //生成随机数
    double speed = 1 / round(random() % 100) + 1.0;
    //对图像视图对象进行设置
    flakeView.frame = CGRectMake(startX, -100.0, 25.0 * scale, 25.0 * scale);
    flakeView.alpha = 0.5; //设置透明度
    [self.view addSubview:flakeView];
    [UIView beginAnimations:nil context:((__bridge void *) (flakeView))];
    [UIView setAnimationDuration:5 * speed]; //设置动画持续时间
    flakeView.frame = CGRectMake(endX, 540.0, 25.0 * scale, 25.0 * scale);
    [UIView commitAnimations];
}
```

【代码解析】

本实例关键功能是雪花的飘落效果。下面就是这个知识点的详细讲解。

在本实例中要想实现自动飘落雪花的效果，首先，需要创建一个定时器，代码如下：

```
[NSTimer scheduledTimerWithTimeInterval:(0.05) target:self selector:
@selector(onTimer) userInfo:nil repeats:YES];
```

在此创建代码中可以看出，定时器会每隔 0.05 秒执行一次 onTimer 方法。在此 onTimer 方法中实现一个雪花的飘落效果，其中需要使用 UIView 块动画，代码如下：

```
[UIView beginAnimations:nil context:((__bridge void *) (flakeView))];
[UIView setAnimationDuration:5 * speed];
flakeView.frame = CGRectMake(endX, 540.0, 25.0 * scale, 25.0 * scale);
[UIView commitAnimations];
```

由于定时器在实行 onTimer 方法时所需的时间间隔很短，所有就会产生大量雪花飘落的效果，至于每一个雪花的飘落的开始位置、结束位置以及速度等的不同，是因为这些值都是随机的，代码如下：


```
int startX = round(random() % 320);
int endX = round(random() % 320);
double scale = 1 / round(random() % 100) + 1.0;
double speed = 1 / round(random() % 100) + 1.0;
```

实例 46 自动旋转的太极

【实例描述】

太极图据说是宋朝道士陈抟所传出，它的内部有黑白两个逗号一样的图形，其中在两个大逗号中间又有两个小圆。本实例实现的功能就是让太极图自动地旋转起来。运行效果如图 4.2 所示。

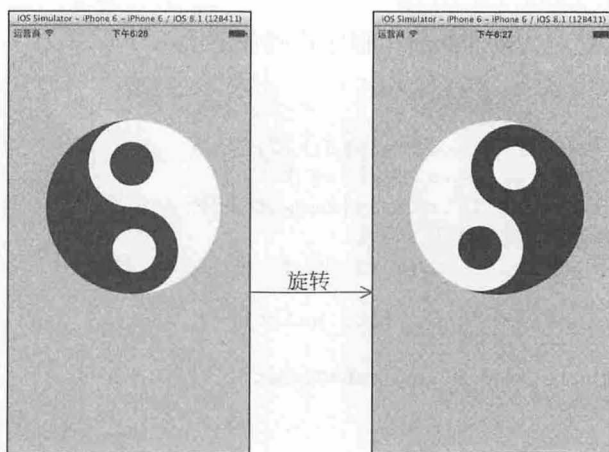


图 4.2 运行效果

【实现过程】

- (1) 创建一个项目，命名为“自动旋转的太极”。
- (2) 创建一个基于 UIView 类的 TaijiView 类。
- (3) 打开 TaijiView.h 文件，编写代码，实现对象以及实例变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface TaijiView : UIView{
    NSTimer* _timer; //声明时间定时器对象
    float currentIndex;
}
@end
```

- (4) 打开 TaijiView.m 文件，编写代码，实现太极的绘制以及自动旋转动画。使用的方法如表 4-1 所示。

表 4-1 TaijiView.m 文件中方法总结

方 法	功 能
initWithFrame:	初始化
updateSpotlight	更新当前索引以及调用绘制方法
drawRect:	绘制太极

太极的绘制需要使用 `initWithFrame:` 和 `drawRect:` 方法。其中, `initWithFrame:` 方法实现对绘制太极的视图进行初始化设置。程序代码如下:

```
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        currentIndex = 0.0;
        self.backgroundColor = [UIColor clearColor];           //设置背景颜色
        //创建定时器
        _timer = [NSTimer scheduledTimerWithTimeInterval:1.0f/32.0f target:
            self selector:@selector(updateSpotlight) userInfo:nil repeats:YES];
    }
    return self;
}
```

`drawRect:` 方法实现对太极的绘制。程序代码如下:

```
- (void)drawRect:(CGRect)rect
{
    float x = self.frame.size.width/2;
    float y = self.frame.size.height/2;
    float radius = self.frame.size.width/2;                    //获取半径
    //判断当前视图的宽度是否大于高度
    if (self.frame.size.width>self.frame.size.height)
    {
        radius = self.frame.size.height/2;
    }
    float runAngle = M_PI*currentIndex;                        //设置
    //判断
    if (runAngle >= 2*M_PI) {
        runAngle -= 2*M_PI;
    }
    CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
    CGColorRef whiteColor = [[UIColor colorWithRed:1 green:1 blue:1 alpha:1] CGColor];
    CGColorRef blackColor = [[UIColor colorWithRed:0 green:0 blue:0 alpha:1] CGColor];
    //绘制半个白色的圆
    CGContextSetFillColor(context, CGColorGetComponents(whiteColor));
    //设置填充颜色
    CGContextAddArc(context, x, y, radius, 0+runAngle, M_PI+runAngle, 0);
    CGContextClosePath(context);
    CGContextFillPath(context);                                //绘制
    //绘制半个黑色的圆
    CGContextSetFillColor(context, CGColorGetComponents(blackColor));
    CGContextAddArc(context, x, y, radius, M_PI+runAngle, M_PI*2+runAngle,
        0);                                                    //添加弧
    CGContextClosePath(context);                                //关闭路径
    CGContextFillPath(context);
    //在黑色的半圆上绘制一个白色的小半圆
    CGContextSetFillColor(context, CGColorGetComponents(whiteColor));
    CGContextAddArc(context, x+radius/2*cos(runAngle), y+radius/2*sin(runAngle), radius/2, M_PI+runAngle, M_PI*2+runAngle, 0); //添加弧
    CGContextClosePath(context);
    CGContextFillPath(context);
    //在白色的半圆上绘制一个黑色的小半圆
```

```

CGContextSetFillColor(context, CGColorGetComponents(blackColor));
CGContextAddArc(context, x-radius/2*cos(runAngle), y-radius/2*sin(run
Angle), radius/2, 0+runAngle, M_PI+runAngle, 0);          //添加弧
CGContextClosePath(context);
CGContextFillPath(context);
//设置白色的线
CGContextSetStrokeColorWithColor(context, whiteColor); //设置线的颜色
CGContextMoveToPoint(context, x+radius*cos(runAngle), y+radius*sin
(runAngle));
CGContextAddLineToPoint(context, x, y);                    //设置结束点
CGContextStrokePath(context);
//设置黑色的线
CGContextSetStrokeColorWithColor(context, blackColor); //设置线的颜色
CGContextMoveToPoint(context, x-radius*cos(runAngle), y-radius*sin
(runAngle));
CGContextAddLineToPoint(context, x, y);
CGContextStrokePath(context);                              //绘制
//绘制白色的圆
CGContextSetFillColor(context, CGColorGetComponents( whiteColor));
CGContextAddArc(context, x-radius/2*cos(runAngle), y-radius/2*sin(run
Angle), radius/4, 0, M_PI*2, 0);                          //添加弧
CGContextClosePath(context);
CGContextFillPath(context);
//绘制黑色的圆
CGContextSetFillColor(context, CGColorGetComponents( blackColor));
CGContextAddArc(context, x+radius/2*cos(runAngle), y+radius/2*sin(run
Angle), radius/4, 0, M_PI*2, 0);
CGContextClosePath(context);                                //关闭路径
CGContextFillPath(context);
}

```

太极的自动旋转动画需要使用 `updateSpotlight` 方法实现。程序代码如下：

```

-(void)updateSpotlight
{
    currentIndex+= 0.01;
    [self setNeedsDisplay];          //重新绘制
}

```

(5) 打开 `ViewController.h` 文件，编写代码，实现头文件以及对象的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "TaijiView.h"
@interface ViewController : UIViewController{
    TaijiView *taiji;
}
@end

```

(6) 打开 `Main.storyboard` 文件，将设计界面的颜色设置为浅灰色。

(7) 打开 `ViewController.m` 文件，编写代码，实现视图对象的创建以及添加。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
}

```

```
// Do any additional setup after loading the view, typically from a nib.
taiji = [[TaijiView alloc] initWithFrame:CGRectMake(50.0f, 80.0f, 230.0f,
320.0f)]; //实例化对象
[self.view addSubview:taiji];
}
```

【代码解析】

本实例关键功能是太极图的绘制以及自动旋转功能。下面依次讲解这两个知识点。

1. 太极图的绘制

在本实例中，可以看到太极图其实就是由多个半圆以及两个圆组成。为了方便，半圆以及圆都可以使用绘制弧线的方法 `CGContextAddArc` 来进行绘制。代码如下：

```
CGContextSetFillColor(context, CGColorGetComponents(whiteColor));
CGContextAddArc(context, x, y, radius, 0+runAngle, M_PI+runAngle, 0);
CGContextClosePath(context);
CGContextFillPath(context);
```

2. 自动旋转

在本实例中，自动旋转的实现，首先是使用 `scheduledTimerWithTimeInterval:target:selector:userInfo:repeats:` 方法创建一个时间定时器，代码如下：

```
_timer = [NSTimer scheduledTimerWithTimeInterval:1.0f/32.0f target:self
selector:@selector(updateSpotlight) userInfo:nil repeats:YES];
```

在此代码中可以看到，在每隔 `1.0f/32.0f` 秒，就会调用 `updateSpotlight` 方法。在 `updateSpotlight` 方法中会对 `currentIndex` 实例变量进行赋值，然后再对太极进行重绘，此时的 `currentIndex` 就是绘制太极的关键参数。

实例 47 礼花效果

【实例描述】

`CAEmitterLayer` 提供了一个基于 Core Animation 的粒子发射系统，使用它可以实现各种各样的动画效果。本实例实现的功能就是使用了 `CAEmitterLayer` 实现了礼花的效果。运行效果如图 4.3 所示。

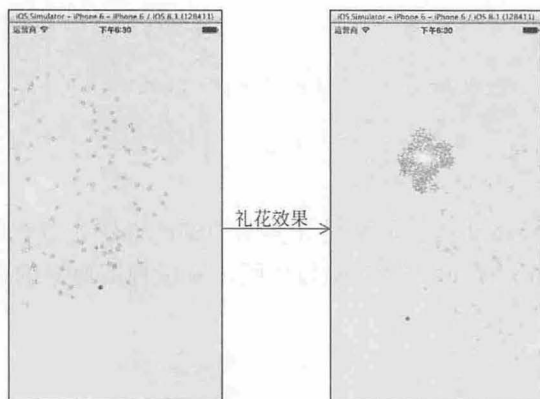


图 4.3 运行效果

【实现过程】

- (1) 创建一个项目，命名为“礼花效果”。
- (2) 添加图像 1.png、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，添加 QuartzCore/CoreAnimation.h 头文件。
- (4) 打开 ViewController.m 文件，编写代码，实现礼花的效果。程序代码如下：

```

- (void) viewDidLoad
{
    [super viewDidLoad];
    CAEmitterLayer *fireworksEmitter = [CAEmitterLayer layer];
    CGRect viewBounds = self.view.layer.bounds;
    //设置发射位置
    fireworksEmitter.emitterPosition = CGPointMake(viewBounds.size.
width/2.0, viewBounds.size.height);
    fireworksEmitter.emitterSize= CGSizeMake(viewBounds.size.width/2.0,
0.0); //发射源的尺寸大小
    fireworksEmitter.emitterMode= kCAEmitterLayerOutline; //发射模式
    fireworksEmitter.emitterShape= kCAEmitterLayerLine; //发射形状
    fireworksEmitter.renderMode= kCAEmitterLayerAdditive; //渲染模式
    fireworksEmitter.seed = (arc4random()%100)+1; //用于初始化随机数函数的种子
    CAEmitterCell* rocket = [CAEmitterCell emitterCell];
    rocket.birthRate= 1.0; //粒子参数的速度乘数因子
    rocket.emissionRange= 0.25 * M_PI; //周围发射角度
    rocket.velocity= 380; //速度
    rocket.velocityRange = 100; //速度范围
    rocket.yAcceleration = 75; //粒子 y 方向的加速度分量
    rocket.lifetime = 1.02;
    //是个 CGImageRef 的对象，即粒子要展现的图片
    rocket.contents = (id) [[UIImage imageNamed:@"1"] CGImage];
    rocket.scale= 0.2; //缩放
    rocket.color= [[UIColor redColor] CGColor]; //粒子的颜色
    rocket.greenRange= 1.0; //一个粒子的绿颜色能改变的范围
    rocket.redRange= 1.0; //一个粒子的红颜色能改变的范围
    rocket.blueRange= 1.0; //一个粒子的蓝颜色能改变的范围
    rocket.spinRange= M_PI; //粒子旋转角度范围
    CAEmitterCell* burst = [CAEmitterCell emitterCell];
    burst.birthRate = 1.0;
    burst.velocity= 0;
    burst.scale= 2.5; //缩放
    burst.redSpeed =-1.5;
    burst.blueSpeed=+1.5;
    burst.greenSpeed=+1.0;
    burst.lifetime= 0.35; //生命周期范围
    CAEmitterCell* spark = [CAEmitterCell emitterCell];
    spark.birthRate = 400;
    spark.velocity= 125; //速度
    spark.emissionRange= 2* M_PI;
    spark.yAcceleration = 75;
    spark.lifetime= 3; //生命周期范围
    spark.contents = (id) [[UIImage imageNamed:@"2"] CGImage]; //粒子的内容
    spark.scaleSpeed =-0.2;
    spark.greenSpeed =-0.1; //粒子绿色在生命周期内的改变速度
}

```



```

    spark.redSpeed = 0.4;
    spark.blueSpeed = -0.1;
    spark.alphaSpeed = -0.25;           //粒子透明度在生命周期内的改变速度
    spark.spin = 2 * M_PI;
    spark.spinRange = 2 * M_PI;
    fireworksEmitter.emitterCells = [NSArray arrayWithObject:rocket];
    rocket.emitterCells = [NSArray arrayWithObject:burst]; //实例化数组
    burst.emitterCells = [NSArray arrayWithObject:spark];
    [self.view.layer addSublayer:fireworksEmitter];
                                     //将 fireworksEmitter 添加到图层上
}

```

【代码解析】

本实例关键功能是礼花效果的实现。下面就是这个知识点的详细讲解。

在本实例中礼花效果的实现需要使用粒子效果。具体步骤如下：

(1) 创建粒子发射系统。CAEmitterLayer 提供了一个基于 Core Animation 的粒子发射系统。它的创建需要使用 layer 方法。在本实例中的代码如下：

```
CAEmitterLayer *fireworksEmitter = [CAEmitterLayer layer];
```

(2) 创建粒子。CAEmitterCell 类是 CAEmitterLayer 发射器用来发射的粒子。其创建需要使用 CAEmitterCell 类的 emitterCell 方法实现，其语法形式如下：

```
+ (id)emitterCell;
```

在本实例中就是使用了 emitterCell 方法对粒子对象进行的创建。代码如下：

```
CAEmitterCell* rocket = [CAEmitterCell emitterCell];
```

(3) 添加粒子。最后需要将粒子添加到 CAEmitterLayer 发射器中用来发射。需要使用 CAEmitterLayer 的 emitterCells:方法实现。其语法形式如下：

```
@property(copy) NSArray *emitterCells;
```

在本实例中就使用了 emitterCells:方法将创建的粒子添加到了创建的 CAEmitterLayer 对象中。代码如下：

```
fireworksEmitter.emitterCells = [NSArray arrayWithObject:rocket];
```

实例 48 物理引擎——掉落的蘑菇

【实例描述】

在 iOS 7 中新增了很多的特性，其中一种就是 UIKit 动力学。UIKit 支持 5 种动力行为，本实例就使用了 UIKit 支持的 UIGravityBehavior 重力行为和 UICollisionBehavior 碰撞行为实现了掉落的蘑菇。当用户轻触界面，蘑菇就会出现并且自由下落，遇到边界或者其他蘑菇时就会出现碰撞行为。运行效果如图 4.4 所示。

【实现过程】

- (1) 创建一个项目，命名为“掉落的蘑菇”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。

(3) 打开 ViewController.h 文件, 编写代码, 实现属性的声明。程序代码如下:

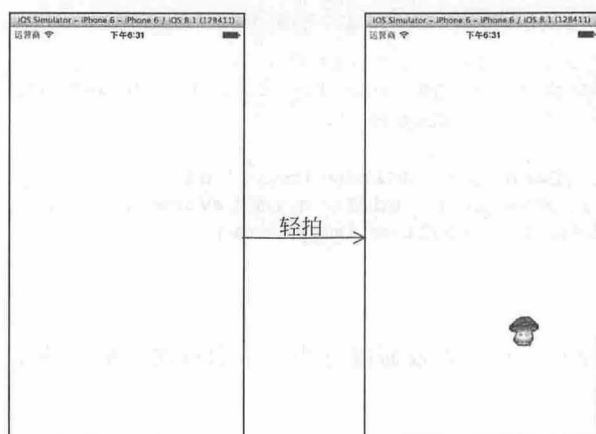


图 4.4 运行效果

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController
//属性
@property (nonatomic, strong) UIDynamicAnimator *animator;
@property (nonatomic, strong) UIGravityBehavior *gravityBehavior;
@property (nonatomic, strong) UICollisionBehavior *collisionBehavior;
@property (nonatomic, strong) UIDynamicItemBehavior *itemBehavior;
@end
```

(4) 打开 ViewController.m 文件, 编写代码, 实现掉落蘑菇的功能。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //创建轻拍手势识别器对象
    UITapGestureRecognizer *gesture = [[UITapGestureRecognizer alloc]
    initWithTarget:self action:@selector(tapped:)];
    [self.view addGestureRecognizer:gesture];
    self.animator = [[UIDynamicAnimator alloc] initWithReferenceView:
    self.view]; //创建动力行为对象
    self.gravityBehavior = [[UIGravityBehavior alloc] initWithItems:nil];
    //创建重力行为对象

    //创建并设置碰撞行为对象
    self.collisionBehavior = [[UICollisionBehavior alloc] initWithItems:nil];
    self.collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;
    //创建并设置动力学元素行为对象
    self.itemBehavior = [[UIDynamicItemBehavior alloc] initWithItems:nil];
    self.itemBehavior.elasticity = 0.6; //弹性系数
    self.itemBehavior.friction = 0.5; //摩擦系数
    self.itemBehavior.resistance = 0.5; //阻力
    //添加动力行为
    [self.animator addBehavior:self.gravityBehavior];
    [self.animator addBehavior:self.collisionBehavior];
    [self.animator addBehavior:self.itemBehavior];
}
//轻拍的实现
```

```

- (void)tapped:(UITapGestureRecognizer *)gesture {
    UIImage *image = [UIImage imageNamed:@"1.png"];
    UIImageView *imageView = [[UIImageView alloc] initWithImage:image];
    [self.view addSubview:imageView]; //添加视图对象
    CGPoint tappedPos = [gesture locationInView:gesture.view];
    imageView.center = tappedPos; //添加中心位置
    //添加条目
    [self.gravityBehavior addItem:imageView];
    [self.collisionBehavior addItem:imageView];
    [self.itemBehavior addItem:imageView];
}

```

【代码解析】

本实例关键功能是重力行为以及碰撞行为。下面依次讲解这两个知识点。

1. 重力行为

在 iOS 7 中，新增了 UIKit 动力学。其中，`UIGravityBehavior` 类表示的就是重力行为，通过 `UIGravityBehavior` 类给动力指定一个重力矢量，具有重力矢量的动力项，会在重力矢量的方向上一直加速。在本实例中就是使用了 `UIGravityBehavior` 类表示了蘑菇的自由落体运动，它的创建及初始化采用了 `UIGravityBehavior` 的 `initWithItems:` 方法，表示初始化一个具有动力项数组的重力行为，其语法形式如下：

```

- (instancetype)initWithItems:(NSArray *)items;

```

其中，`(NSArray *)items` 表示动力项。在本实例中的代码如下：

```

self.gravityBehavior = [[UIGravityBehavior alloc] initWithItems:nil];

```

其中，`nil` 表示没有动力项。

2. 碰撞行为

`UICollisionBehavior` 类表示的就是碰撞行为，通过对象 `UICollisionBehavior` 指定一个边界，并且让各个动力项在该边界内参与碰撞。在本实例中就是使用了 `UICollisionBehavior` 类指定了重力行为的边界，并且在该边界处参与了碰撞效果，它的创建以及初始化采用了 `UICollisionBehavior` 的 `initWithItems:` 方法，它表示初始化一个具有动力项数组的碰撞行为，代码如下：

```

self.collisionBehavior = [[UICollisionBehavior alloc] initWithItems:nil];

```

其中，对于边界的设置，可以有 4 种方法：

(1) `addBoundaryWithIdentifier:forPath:`

`addBoundaryWithIdentifier:forPath:` 方法实现添加贝塞尔路径作为碰撞行为的边界，其语法形式如下：

```

- (void)addBoundaryWithIdentifier:(id<NSCopying>)identifier forPath:(UIBezierPath *)bezierPath;

```

其中，`(id<NSCopying>)identifier` 表示添加边界的任意标识符；`(UIBezierPath *)bezierPath` 表示贝塞尔路径。

(2) addBoundaryWithIdentifier:fromPoint:toPoint:

addBoundaryWithIdentifier:fromPoint:toPoint:方法实现添加一条线段作为碰撞行为的边界,其语法形式如下:

```
-(void)addBoundaryWithIdentifier:(id<NSCopying>)identifier fromPoint:(CGPoint)p1 toPoint:(CGPoint)p2;
```

其中, (id<NSCopying>)identifier 表示添加边界的任意标识符; (CGPoint)p1 表示边界线段的起点; (CGPoint)p2 表示边界线段的终点。

(3) setTranslatesReferenceBoundsIntoBoundaryWithInsets:

setTranslatesReferenceBoundsIntoBoundaryWithInsets:方法实现设定某一区域作为碰撞的边界,其语法形式如下:

```
-(void)setTranslatesReferenceBoundsIntoBoundaryWithInsets:(UIEdgeInsets) insets;
```

其中, (UIEdgeInsets)insets 表示设置的区域。

(4) translatesReferenceBoundsIntoBoundary

translatesReferenceBoundsIntoBoundary 属性实现的功能是设定某一区域,将此区域作为碰撞的边界,如果为 NO,表示不可以,其语法形式如下:

```
@property(n nonatomic, readwrite) BOOL translatesReferenceBoundsIntoBoundary;
```

在本实例中,对于边界的设定,就是使用了 UICollisionBehavior 的 translatesReferenceBoundsIntoBoundary 属性实现的,代码如下:

```
self.collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;
```

它的边界是针对于用户设置的 animator 中的 referenceView 而定的,即 self.view。

实例 49 物理引擎——橡皮筋

【实例描述】

本实例使用了 UIKit 支持的 UIGravityBehavior 重力行为、UICollisionBehavior 碰撞行为和 UIAttachmentBehavior 吸附行为实现了橡皮筋的功能。当用户轻拍拖动界面的线,和线相连的方块就会运动,并且在碰到边界时,方块会发生反弹效果。运行效果如图 4.5 所示。

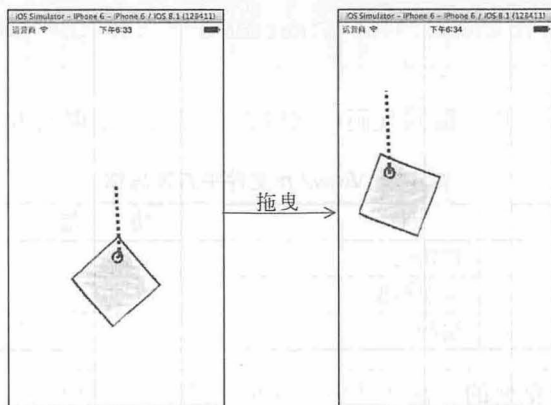


图 4.5 运行效果

【实现过程】

- (1) 创建一个项目，命名为“橡皮筋”。
- (2) 添加图像 1.png、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 View 类。
- (4) 打开 View.h 文件，编写代码，实现属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface View : UIView
//属性
@property (weak, nonatomic) UIImageView *box;
@property (strong, nonatomic) UIDynamicAnimator *animator;
@end
```

- (5) 打开 View.m 文件，编写代码，实现初始化的设置。程序代码如下：

```
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor colorWithPatternImage:[UIImage
            imageNamed:@"1"]];
        //创建并设置图像视图对象
        UIImageView *box = [[UIImageView alloc] initWithImage:[UIImage
            imageNamed:@"2"]];
        box.center = self.center; //设置中心位置
        _box = box;
        [self addSubview:box];
        _animator = [[UIDynamicAnimator alloc] initWithReferenceView:self];
        //创建动力行为对象
    }
    return self;
}
```

- (6) 创建一个基于 View 类的 View2 类。
- (7) 打开 View2.h 文件，编写代码，实现对象以及属性的声明。程序代码如下：

```
#import "View.h"
@interface View2 : View{
    UIImageView *_imageView;
}
@property (strong, nonatomic) UIAttachmentBehavior *attachment;
@end
```

- (8) 打开 View2.m 文件，编写代码，实现附着行为。使用的方法如表 4-2 所示。

表 4-2 View2.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
pan:	实现拖曳
drawRect:	绘制

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，initWithFrame: 方法实现一些设置的初始化功能。程序代码如下：


```

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        //创建并添加拖曳手势识别器对象
        UIPanGestureRecognizer *pan = [[UIPanGestureRecognizer alloc]
            initWithTarget:self action:@selector(pan:)]; //创建拖曳手势识别器对象
        [self addGestureRecognizer:pan];
        //添加附着行为
        UIOffset offset = UIOffsetMake(-25, -25);
        attachment = [[UIAttachmentBehavior alloc] initWithItem:self.box
            offsetFromCenter:offset attachedToAnchor:CGPointMake (self.box.
            center.x, self.box.center.y - 100)]; //创建吸附行为对象
        [self.animator addBehavior:attachment];
        //添加 box 内部锚点图像
        UIImageView *imageView = [[UIImageView alloc] initWithImage:
            [UIImage imageNamed:@"3"]];
        imageView.center = CGPointMake(self.box.bounds.size.width / 2 +
            offset.horizontal, self.box.bounds.size.height / 2 + offset.vertical);
            //设置中心位置
        [self.box addSubview:imageView];
        _imageView = imageView;
    }
    return self;
}

```

drawRect:方法实现对线的绘制。程序代码如下:

```

- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGPoint p = [self convertPoint:_imageView.center fromView:self.box];
    CGContextMoveToPoint(context, p.x, p.y); //设置开始点
    CGContextAddLineToPoint(context, attachment.anchorPoint.x, attachment.
        anchorPoint.y);
    CGContextSetLineWidth(context, 5.0f); //设置线宽
    CGFloat length[] = {5.0, 5.0};
    CGContextSetLineDash(context, 0.0, length, 2);
    [[UIColor blackColor] set];
    CGContextDrawPath(context, kCGPathStroke); //绘制路径
}

```

(9) 创建一个基于 View2 类的 View3 类。

(10) 打开 View3.m 文件, 编写代码, 实现橡皮筋的弹性行为。程序代码如下:

```

- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.attachment.damping = 1;
        self.attachment.frequency = 1;
        //设置观察者, 在手指离开以后继续画线
        [self.box addObserver:self forKeyPath:@"center" options:NSKeyValue
            Observing OptionNew context:nil];
        //创建并添加重力
        UIGravityBehavior *gravity = [[UIGravityBehavior alloc] initWith
            Items:@[self.box]];
    }
}

```

```

        [self.attachment addChildBehavior:gravity];
        //创建并添加碰撞行为对象
        UICollisionBehavior *collision = [[UICollisionBehavior alloc]
            initWithItems:@[self.box]];
        collision.translatesReferenceBoundsIntoBoundary = YES;
        [self.attachment addChildBehavior:collision];
    }
    return self;
}
//画线
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary *)change context:(void *)context {
    [self setNeedsDisplay]; //重新绘制
}

```

(11) 打开 ViewController.h 文件，编写代码，实现头文件以及对象的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "View.h"
#import "View3.h"
@interface ViewController : UIViewController{
    View *vv;
}
@end

```

(12) 打开 ViewController.m 文件，编写代码，实现对对象的创建以及添加。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    vv=[[View3 alloc] initWithFrame:CGRectMake(0, 0, 320, 680)]; //实例化对象
    [self.view addSubview:vv];
}

```

【代码解析】

本实例关键功能是吸附行为。下面就是这个知识点的详细讲解。

UIAttachmentBehavior 类表示的就是吸附行为，用来指定两个动力项（项或点）之间的连接。当一个项或者点移动时，吸附的项也随之移动。如果想使用它，首先要创建其对象。在本实例中使用了 UIAttachmentBehavior 的 initWithItem:offsetFromCenter:attachedToAnchor:方法，初始化连接动力项中某一点和锚点的吸附行为，其语法形式如下：

```

- (instancetype)initWithItem:(id<UIDynamicItem>)item offsetFromCenter:
(UIOffset)p1 attachedToAnchor:
(CGPoint)point;

```

其中，(id<UIDynamicItem>)item 表示吸附行为的动力项；(UIOffset)p1 表示对于动力项中心的偏移；(CGPoint)point 表示吸附行为的锚点。在实例中的创建 UIAttachmentBehavior 对象的代码如下：

```

attachment = [[UIAttachmentBehavior alloc] initWithItem:self.box
offsetFromCenter:offset attachedToAnchor:CGPointMake(self.box.center.x,
self.box.center.y - 100)];

```

其中, `self.box` 表示吸附行为的动力项; `offset` 表示对于动力项中心的偏移; `CGPointMake(self.box.center.x, self.box.center.y - 100)`表示吸附行为的锚点。

实例 50 吃 豆 豆

【实例描述】

本实例实现的功能是一个吃豆豆的小游戏。当用户点击界面上的方向键后,吃豆豆的 PAC-Man 就会移动。当达到某一范围时,豆豆就会被 PAC-Man 吃掉,从而再在别处出现豆豆。运行效果如图 4.6 所示。



图 4.6 运行效果

【实现过程】

- (1) 创建一个项目, 命名为“吃豆豆”。
- (2) 添加图像 1.png~9.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 `ViewController.h` 文件, 编写代码, 实现插座变量以及动作的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet UIImageView *Man;
    IBOutlet UIImageView *dou;
}
//动作
- (IBAction)Up:(id) sender;           //触摸按钮向上移动
- (IBAction)Down:(id) sender;        //触摸按钮向下移动
- (IBAction)Left:(id) sender;         //触摸按钮向左移动
- (IBAction)Right:(id) sender;        //触摸按钮向右移动
@end
```

- (4) 打开 `Main.Storyboard` 文件, 对 `View Controller` 视图控制器的设计界面进行设计, 效果如图 4.7 所示。

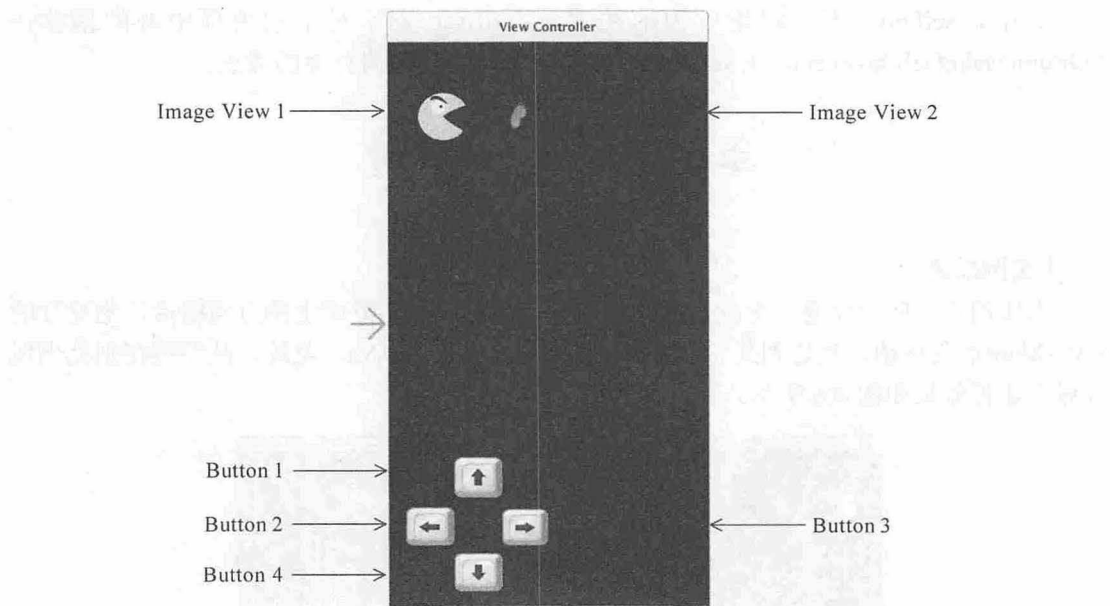


图 4.7 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 4-3 所示。

表 4-3 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 黑色	
Image View1	Image: 1.png	与插座变量 Man 关联
Image View2	Image: 5.png	与插座变量 dou 关联
Button1	Title: (空) Background: 8.png	与动作 Up:关联
Button2	Title: (空) Background: 6.png	与动作 Left:关联
Button3	Title: (空) Background: 9.png	与动作 Right:关联
Button4	Title: (空) Background: 7.png	与动作 Down:关联

(5) 打开 ViewController.m 文件，编写代码，实现吃豆豆的功能。使用的方法如表 4-4 所示。

表 4-4 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
Up:	单击向上按钮
Down:	单击向下按钮
Left:	单击向左按钮
Right:	单击向右按钮

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewDidLoad方法实现对图像视图中心位置的设定。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    int i=arc4random()%4;           //产生随机数
    switch (i) {
        case 0:
            dou.hidden=NO;
            dou.center=CGPointMake(50, 100);           //设置中心位置
            break;
        case 1:
            dou.hidden=NO;
            dou.center=CGPointMake(200, 150);           //设置中心位置
            break;
        case 2:
            dou.hidden=NO;
            dou.center=CGPointMake(250, 200);           //设置中心位置
            break;
        default:
            dou.hidden=NO;
            dou.center=CGPointMake(150, 300);           //设置中心位置
            break;
    }
}
```

Up:方法实现单击向上的按钮后，实现图像的移动。程序代码如下：

```
- (IBAction)Up:(id)sender {
    if (Man.center.y>50) {
        [UIImageView beginAnimations:nil context:((__bridge void *) (Man))];
        [UIImageView setAnimationDuration:0.5];           //设置动画持续时间
        Man.center=CGPointMake(Man.center.x, Man.center.y-10);
        Man.image=[UIImage imageNamed:@"3.png"];           //设置图像
        if (Man.center.x-20<dou.center.x&&Man.center.x+20>dou.center.x &&
            Man.center.y-15<dou.center.y&&Man.center.y+15>dou.center.y) {
            dou.hidden=YES;           //隐藏豆子的图像
            [self performSelector:@selector(viewDidLoad) withObject:self
                afterDelay:1.0];
        }
        [UIImageView commitAnimations];
    }
}
```

【代码解析】

本实例关键功能是 PAC-Man 的移动以及吃豆豆。下面依次讲解这两个知识点。

1. 物体的移动

在本实例中，PAC-Man 在上、下、左、右方向上的移动都是使用按钮控制的，以向上移动为例，在单击向上的按钮后，PAC-Man 向上移动 10，其中，需要将 PAC-Man 的中心点进行设置，其代码如下：

```
Man.center=CGPointMake(Man.center.x, Man.center.y-10);
```


移动的动画形成使用了 UIView 块动画。

2. 吃豆豆

在 PAC-Man 移动到豆豆面前时，豆豆就会被吃掉即豆豆消失。它的实现，需要为吃豆豆的范围进行设置，如果豆豆在此范围内就会被吃掉，产生下一个豆豆，在本实例中的代码如下：

```
if (Man.center.x-20<dou.center.x&&Man.center.x+20>dou.center.x &&
    Man.center.y-15<dou.center.y&&Man.center.y+15>dou.center.y) {
    dou.hidden=YES;
    [self performSelector:@selector(viewDidLoad) withObject:self after
        Delay:1.0];
}
```

实例 51 打 砖 块

【实例描述】

本实例实现的功能是打砖块的小游戏。当用户单击“开始”按钮，就会出现一个向上运动的小球，当小球碰到砖块时，碰到的砖块就会消失，并且小球会向下掉，必须要使用弹板接住小球，让小球再一次向上运动去碰撞砖块；如果弹板没有接住小球，那么就会弹出一个 Game over 的警告视图。运行效果如图 4.8 所示。

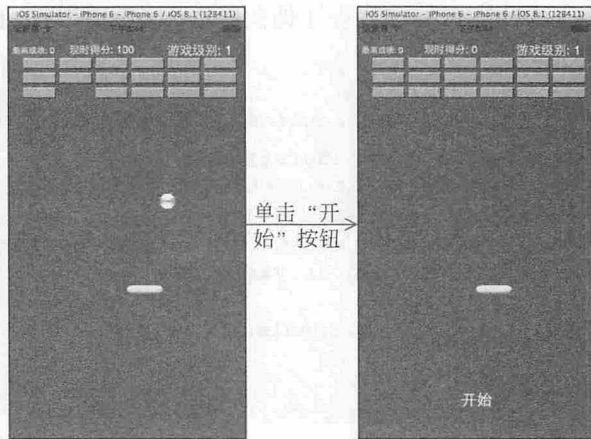


图 4.8 运行效果

【实现过程】

- (1) 创建一个项目，命名为“打砖块”。
- (2) 添加图像 1.png、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIImageView 类的 BoardView 类。
- (4) 打开 BoardView.h 文件，编写代码，实现实例变量的声明。
- (5) 打开 BoardView.m 文件，编写代码，实现触摸功能。程序代码如下：

```
//触摸开始时调用
```

```

- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    //定义图形位置变量对象的坐标, locationInView 表明视图中触摸点的坐标
    startLocation= [[touches anyObject] locationInView:self];
    //把 BoardView 子视图放在最前面
    [[self superview] bringSubviewToFront:self];
}
//获取手指的触摸移动事件方法
- (void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event{
    //定义图形位置变量对象的坐标, locationInView 表明视图中触摸点的坐标
    CGPoint pt=[[touches anyObject] locationInView:self];
    //创建图形框架尺寸变量的对象
    CGRect frame=[self frame];
    //定义图形框架位置变量对象的 x 坐标值, pt.x 表明图形位置变量对象的 x 坐标值,
    startLocation.x 表明触摸点的 x 坐标
    frame.origin.x = frame.origin.x + (pt.x-startLocation.x);
    //设定图形框架位置变量数值
    [self setFrame:frame];
}

```

(6) 打开 ViewController.h 文件, 编写代码, 实现头文件、宏定义、插座变量、对象、实例变量、属性以及方法的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "BoardView.h"
//宏定义
#define BRICKHEIGHT 15
#define BRICKWIDTH 44
#define BOARDHEIGHT 10
#define BOARDWIDTH 48
#define TOP 40
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet UILabel *highestLabel;
    IBOutlet UILabel *levelLabel;
    IBOutlet UILabel *scoreLabel;
    IBOutlet UIButton *button;
    //对象
    NSTimer *timer;
    UIImageView *ball;
    BoardView *board;
    NSMutableArray *bricks;
    //实例变量
    int level,numOfBricks,score,highest;
    double speed;
    CGPoint moveDis;
}
//属性
@property int level,score,highest;
@property int numOfBricks;
@property double speed;
//动作
- (IBAction)Play:(id)sender;
- (void)levelMap:(int)inlevel;
@end

```

(7) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计,

效果如图 4.9 所示。

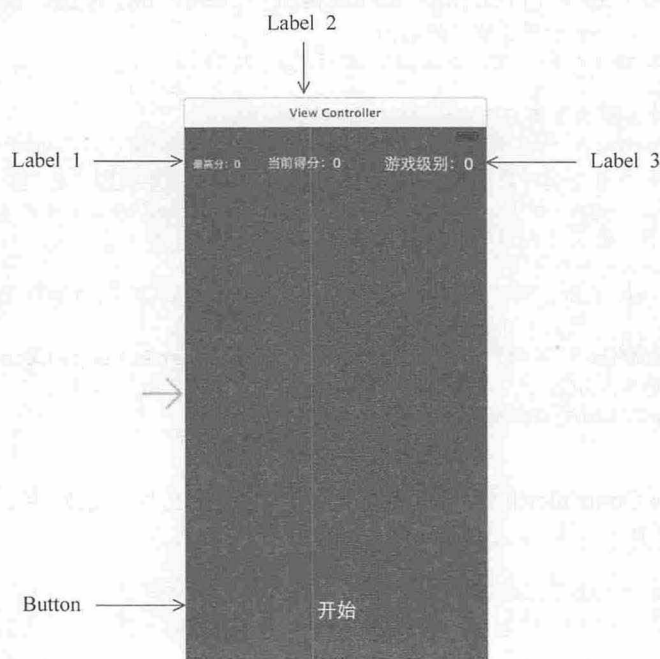


图 4.9 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 4-5 所示。

表 4-5 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 深灰色	
Label1	Text: 最高分:0 Color: 白色 Font: System 11.0 Alignment: 居中	与插座变量 highestLabel 关联
Label2	Text: 当前得分:0 Color: 白色 Font: System 14.0 Alignment: 居中	与插座变量 scoreLabel 关联
Label3	Text: 游戏级别:0 Color: 白色 Font: System 17.0 Alignment: 居中	与插座变量 levelLabel 关联
Button	Title: 开始 Text Color: 白色	与插座变量 button 关联 与动作 Play:关联

(8) 打开 ViewController.m 文件，编写代码，实现打砖块的功能。使用的方法如表 4-6 所示。

表 4-6 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
levelMap:	显示游戏的级别
onTimer	定时器实现的方法
Play:	单击“开始”按钮
alertView.clickedButtonAtIndex:	实现警告视图的响应

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, viewDidLoad方法在视图加载后调用, 实现初始化的功能。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //球的动态移动距离
    moveDis=CGPointMake(-3, -3);
    speed=0.03;
    board=[[BoardView alloc] initWithImage:[UIImage imageNamed:@"2.png"]];
    [board setUserInteractionEnabled:YES];
    //创建并设置图像视图对象, 用来作弹板
    board.frame=CGRectMake(160, 360, BOARDWIDTH, BOARDHEIGHT); //设置框架
    ball=[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"3.png"]];
    [self.view addSubview:board];
    level=1;
    score=0;
    highest=0;
    设置标签内容
    levelLabel.text=[NSString stringWithFormat:@"游戏级别: %i", level];
    scoreLabel.text=[NSString stringWithFormat:@"显示得分: %i", score];
    highestLabel.text=[NSString stringWithFormat:@"最高成绩: %i", highest];
    [self levelMap:level];
}

```

levelMap:方法实现显示游戏的级别即对“砖块”进行创建以及设置。程序代码如下:

```

- (void)levelMap:(int)inlevel {
    //创建图像视图对象, 用来显示“砖块”
    UIImageView *brick;
    switch (inlevel) {
        case 1:
            bricks=[NSMutableArray arrayWithCapacity:20]; //创建可变数组
            //砖块数量
            numOfBricks=20;
            for (int i=0; i<3; i++) {
                for (int j=0; j<6; j++) {
                    //实例化图像对象, 用来显示砖块
                    brick=[[UIImageView alloc] initWithImage:[UIImage
                        imageNamed:@"1.png"]];
                    brick.frame=CGRectMake(20+j*BRICKWIDTH+j*5, TOP+10+
                        BRICKHEIGHT*i+5*i, BRICKWIDTH, BRICKHEIGHT); //设置框架
                    [self.view addSubview:brick];
                    [bricks addObject:brick];
                }
            }
        }
    }
}

```

```

        [bricks addObject:brick]; //添加对象
        break;
    case 2:
        bricks=[NSMutableArray arrayWithCapacity:28];
        numOfBricks=20; //砖块数量
        for (int i=0; i<7; i++) {
            for (int j=0;j<2;j++) {
                //实例化图像对象，用来显示砖块
                brick=[[UIImageView alloc] initWithImage:[UIImage
                    imageNamed:@"1.png"]];
                brick.frame=CGRectMake(20+j*BRICKWIDTH+j*5, TOP+10+
                    BRICKHIGHT*i+5*i, BRICKWIDTH, BRICKHIGHT); //设置框架
                [self.view addSubview:brick];
                [bricks addObject:brick];
            }
        }
        for (int i=0; i<7; i++) {
            for (int j=0;j<2;j++) {
                //实例化图像对象，用来显示砖块
                brick=[[UIImageView alloc] initWithImage:[UIImage
                    imageNamed:@"1.png"]];
                brick.frame=CGRectMake(20+j*BRICKWIDTH+j*5+180, TOP+
                    10+ BRICKHIGHT*i+5*i, BRICKWIDTH, BRICKHIGHT);
                //设置框架
                [self.view addSubview:brick];
                [bricks addObject:brick];
            }
        }
        break;
    default:
        break;
}
}

```

onTimer 方法实现在定时器中实现的方法，它主要使用打砖块的功能。程序代码如下：

```

- (void)onTimer{
    float posx,posy;
    //球的中心坐标
    posx=ball.center.x; posy=ball.center.y;
    ball.center = CGPointMake(posx+moveDis.x, posy+moveDis.y);
    //创建“球”图像视图对象的中心
    //判断球的中心的 x 值是否大于 305 小于 15
    if (ball.center.x>305 || ball.center.x<15 ) {
        moveDis.x=-moveDis.x;
    }
    //判断球的中心的有 y 值是否小于 TOP
    if ( ball.center.y<TOP ) {
        moveDis.y=-moveDis.y;
    }
    int j=[bricks count]; //获取“砖块”的总数
    for (int i=0; i<j; i++) {
        UIImageView *brick=(UIImageView *)[bricks objectAtIndex:i];
        //实例化对象
        if (CGRectIntersectsRect(ball.frame, brick.frame)&&[brick

```



```

superview)) {
    score+=100;
    [brick removeFromSuperview]; //移除指定的视图
    //判断产生的随机数是否为5的倍数
    if (rand()%5==0) {
        UIImageView* imageView = [[UIImageView alloc] initWithImage:
            [UIImage imageNamed:@"4.png"]]; //实例化对象
        imageView.frame = CGRectMake(brick.frame.origin.x,
            brick.frame.origin.y, 48, 48);
        [self.view addSubview:imageView];
        //动画效果
        [UIView beginAnimations:nil context:(__bridge void
            *) (imageView)];
        [UIView setAnimationDuration:5.0]; //设置动画持续时间
        [UIView setAnimationCurve:UIViewAnimationCurveEaseOut];
        imageView.frame = CGRectMake(brick.frame.origin.x, 380, 40,
            40); //设置框架
        [UIView setAnimationDelegate:self]; //设置动画的委托
        [UIView commitAnimations];
    }
    numOfBricks--;
    if ((ball.center.y-16<brick.frame.origin.y+BRICKHEIGHT || ball.
        center.y+16>brick.frame.origin.y)
        && ball.center.x>brick.frame.origin.x && ball.center.
        x<brick.frame.origin.x+BRICKWIDTH) {
        moveDis.y=-moveDis.y; //反向移动
    }else if (ball.center.y>brick.frame.origin.y && ball.center.
        y<brick.frame.origin.y+BRICKHEIGHT&&(ball.center.x+16>brick.
        frame.origin.x || ball.center.x-16<brick.frame.origin.
        x+BRICKWIDTH)){
        moveDis.x=-moveDis.x; //反向移动
    }else{
        moveDis.x=-moveDis.x;
        moveDis.y=-moveDis.y;
    }
    break;
}
}
//判断“砖块”的数量是否为0
if (numOfBricks==0) {
    if (level<2) {
        [ball removeFromSuperview]; //移除球
        [timer invalidate]; //让定时器失效
        level++; speed=speed-0.003;
        levelLabel.text=[NSString stringWithFormat:@"Level %i",level];
        //设置文本内容
        [self levelMap:level];
    }else{
        UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"K.O."
            message:@"恭喜! 你赢了" delegate:self cancelButtonTitle:
            @"OK" otherButtonTitles:nil]; //创建警告视图
        [alert show];
    }
}
//当“球”图像视图对象的位置和“弹板”接触时

```

```

if (CGRectIntersectsRect(ball.frame, board.frame)) {
    if(ball.center.x>board.frame.origin.x&&ball.center.x<board.frame.
        origin.x+BOARDWIDTH) {
        //反向移动
        moveDis.y=-moveDis.y;
    }else {
        moveDis.x=-moveDis.x;
        moveDis.y=-moveDis.y;
    }
}
}else{
    //判断“球”图像视图对象的中心是否超出了屏幕坐标位置 y 值
    if (ball.center.y>380){
        [ball removeFromSuperview];
        [timer invalidate]; //让定时器失效
        timer=NULL;
        UIAlertView *alert=[[UIAlertView alloc] initWithTitle:@"Game
            over"message:@"你输了, 继续获取更好的成绩..." delegate:self
            cancelButtonTitle:@"确定"otherButtonTitles:nil]; //创建警告视图
        [alert show];
    }
}
scoreLabel.text=[NSString stringWithFormat:@"现时得分: %i",score];
//设置文本内容
}

```

Play:方法实现单击“开始”按钮后, 开始游戏。程序代码如下:

```

- (IBAction)Play:(id)sender {
    button.hidden=YES;
    if(!timer){
        timer=[NSTimer scheduledTimerWithTimeInterval:speed target:self
            selector:@selector(onTimer) userInfo:nil repeats:YES];
        //创建警告视图

        //球开始时的坐标
        ball.frame=CGRectMake(160, 328, 20, 20);
        [self.view addSubview:ball];
        //弹板开始时的坐标
        board.frame=CGRectMake(160, 360, 48, 10);
    }
}

```

【代码解析】

本实例关键功能是将球碰到界面、弹板以及砖块的反弹。下面就是这个知识点的详细讲解。

以球碰到界面的反弹为例, 在本实例中它的实现需要使用球的中心和界面的边界去判断, 代码如下:

```

if (ball.center.x>305 || ball.center.x<15 ) {
    moveDis.x=-moveDis.x;
}
if ( ball.center.y<TOP ) {
    moveDis.y=-moveDis.y;
}

```

实例 52 碰撞的火球

【实例描述】

本实例实现一个碰撞的火球。当获取碰到界面后就会实现反弹的动作，并且火球的阴影会慢慢地变小以及消失。运行效果如图 4.10 所示。

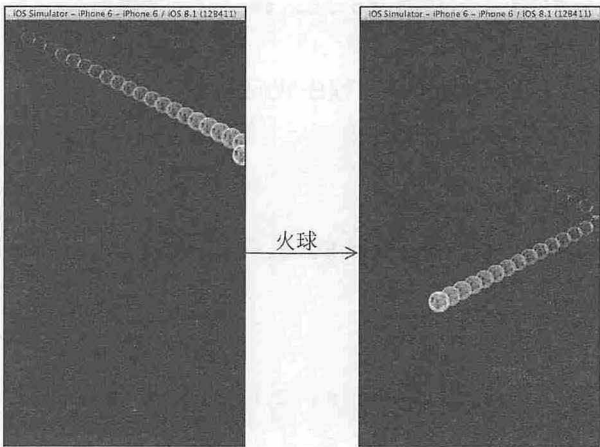


图 4.10 运行效果

【实现过程】

- (1) 创建一个项目，命名为“碰撞的火球”。
- (2) 添加图像 1.png、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现实例变量、对象的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    CGPoint pos; //实例变量
    UIImageView* fireBall; //对象
}
@end
```

- (4) 打开 ViewController.m 文件，编写代码，实现碰撞的火球效果。使用的方法如表 4-7 所示。

表 4-7 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
removeSmoke:finished:context:	图像视图的移除
onTimer	定时器的方法，实现效果的反弹动画

viewDidLoad 方法在视图加载后调用，实现初始化功能。程序代码如下：

```
- (void)viewDidLoad
```

```

{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    pos = CGPointMake(15.0, 7.5);
    self.view.backgroundColor = [UIColor blackColor];
    //创建并设置图像视图对象，用作球
    fireBall = [[UIImageView alloc] initWithImage:[UIImage imageNamed:
@"1.png"]];
    fireBall.frame = CGRectMake(0, 0, 32, 32);
    [self.view addSubview:fireBall];
    [NSTimer scheduledTimerWithTimeInterval:(0.05) target:self selector:
@selector(onTimer) userInfo:nil repeats:YES];    //创建定时器
}

```

onTimer 方法实现火球的碰撞效果。程序代码如下：

```

-(void) onTimer {
    //获取图像视图中心点
    float x = fireBall.center.x;
    float y = fireBall.center.y;
    fireBall.center = CGPointMake(fireBall.center.x + pos.x, fireBall.
center.y + pos.y);
    //判断边界
    if (fireBall.center.x > 320 || fireBall.center.x < 0)
        pos.x = -pos.x;    //反向移动
    if (fireBall.center.y > 460 || fireBall.center.y < 0)
        pos.y = -pos.y;    //反向移动
    //创建并设置图像视图对象，用作球的轨迹
    UIImageView* imageView = [[UIImageView alloc] initWithImage:[UIImage
imageNamed:@"2.png"]];
    imageView.frame = CGRectMake(x-16, y-16, 32, 32);    //设置框架
    [self.view addSubview:imageView];
    //“轨迹”图像视图对象的动画效果
    [UIView beginAnimations:nil context:((__bridge void *) (imageView))];
    [UIView setAnimationDuration:3.0];    //设置动画持续时间
    [UIView setAnimationCurve:UIViewAnimationCurveEaseOut];
    imageView.frame = CGRectMake(x-6, y-6, 12, 12);
    [imageView setAlpha:0.0];    //设置透明度
    [UIView setAnimationDelegate:self];    //设置动画的委托
    [UIView setAnimationDidStopSelector:@selector(removeSmoke: finished:
context:)];
    [UIView commitAnimations];
    [self.view bringSubviewToFront:fireBall];
}

```

【代码解析】

本实例关键功能是获取阴影的动画效果。下面就是这个知识点的详细讲解。

在本实例中，火球的阴影会慢慢地变小最终消失，它的实现需要使用 UIView 的块动画，代码如下所示：

```

[UIView beginAnimations:nil context:((__bridge void *) (imageView))];
[UIView setAnimationDuration:3.0];
[UIView setAnimationCurve:UIViewAnimationCurveEaseOut];
imageView.frame = CGRectMake(x-6, y-6, 12, 12);
[imageView setAlpha:0.0];
.....
[UIView commitAnimations];

```


实例 53 旋转的滚珠

【实例描述】

本实例实现的功能是一个旋转的滚珠。当单击“运行”按钮后，首先显示的是滚珠以矩阵的方式有序排列，当用户轻拍界面后，圆珠就会以一个圆环的形式排列，并且此圆环可以旋转以及缩放。运行效果如图 4.11 所示。

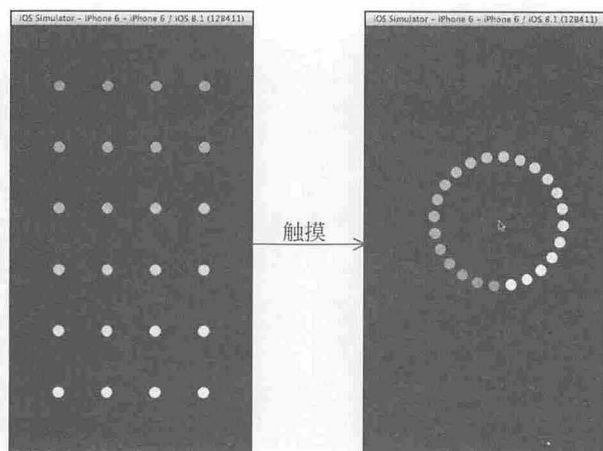


图 4.11 运行效果

【实现过程】

- (1) 创建一个项目，命名为“旋转的滚珠”。
- (2) 创建一个基于 CALayer 类的 Layer 类。
- (3) 打开 Layer.h 文件，编写代码，实现对象以及属性的声明。程序代码如下：

```
#import <QuartzCore/QuartzCore.h>
@interface Layer : CALayer{
    UIColor *color;
}
@property (retain, nonatomic) UIColor *color;
@end
```

- (4) 打开 Layer.m 文件，编写代码，实现对圆的绘制。使用的方法如表 4-8 所示。

表 4-8 Layer.m 文件中方法总结

方 法	功 能
init	初始化
drawInContext:	绘制
setBounds:	设置边界

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，drawInContext: 方法实现对滚珠的绘制。程序代码如下：

```
- (void)drawInContext:(CGContextRef) ctx
```



```
{
    UIGraphicsPushContext(ctx);
    [color setFill];
    UIBezierPath *path = [UIBezierPath bezierPathWithOvalInRect:self.
    bounds]; //创建贝塞尔路径
    [path fill];
    UIGraphicsPopContext();
}
```

(5) 创建一个基于 UIView 类的 View 类。

(6) 打开 View.h 文件，编写代码，实现头文件、对象、实例变量以及属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "Layer.h" //头文件
@interface View : UIView{
    //对象
    NSMutableArray *layers;
    UITouch *currentTouch;
    NSTimer *rotateTimer;
    //实例变量
    NSUInteger rotateIndex;
    CGFloat radius;
    BOOL animationFlag;
}
@property (retain, nonatomic) UITouch *currentTouch;
@end
```

(7) 打开 View.m 文件，编写代码，实现滚珠的布局以及触摸功能。使用的方法如表 4-9 所示。

表 4-9 View.m文件中方法总结

方 法	功 能
_init	初始化
initWithCoder:	初始化、实例化对象
initWithFrame:	使用框架进行初始化
timerMethod:	定时器方法
layoutSubviews	布局
touchesBegan:withEvent:	开始触摸
touchesMoved:withEvent:	移动触摸
touchesEnded:withEvent:	结束触摸
touchesCancelled:withEvent:	取消触摸

滚珠的布局需要使用_init、initWithCoder:、initWithFrame:、layoutSubviews 方法。其中，_init 方法实现一些初始化的设置。程序代码如下：

```
- (void)_init
{
    radius = 80.0;
    self.backgroundColor = [UIColor blackColor];
    //设置背景
    layers = [[NSMutableArray alloc] init];
}
```

```

NSUInteger count = 24;
//创建圆珠
for (int i = 0; i < count; i++) {
    Layer *aLayer = [Layer layer];
    aLayer.color = [UIColor colorWithWhite:(0.5 + 0.5 * ((float)i/count))
    alpha:1.0];
    aLayer.bounds = CGRectMake(0.0, 0.0, 15.0, 15.0); //设置边界
    [layers addObject:aLayer]; //添加对象
}
[self setNeedsLayout];
}

```

layoutSubviews 方法实现对子视图的初始化。程序代码如下：

```

- (void)layoutSubviews
{
    if (self.currentTouch) {
        if (!rotateTimer) {
            //创建定时器对象
            rotateTimer = [NSTimer scheduledTimerWithTimeInterval:0.1
            target:self selector:@selector(timerMethod:) userInfo:nil
            repeats:YES] ;
        }
        CGPoint p = [self.currentTouch locationInView:self]; //获取触摸点
        NSUInteger index = rotateIndex;
        //布局
        for (Layer *aLayer in layers) {
            CGFloat r = M_PI * 2 * index / [layers count];
            CGFloat x = p.x + cos(r) * radius * -1;
            CGFloat y = p.y + sin(r) * radius * -1;
            aLayer.position = CGPointMake(x, y); //设置位置
            index++;
        }
        return;
    }
    radius = 80.0;
    //判断定时器是否不为空
    if (rotateTimer) {
        rotateIndex = 0;
        [rotateTimer invalidate]; //让定时器失效
        rotateTimer = nil;
    }
    NSUInteger index = 0;
    NSUInteger itemsPerRow = 4;
    NSUInteger rows = (NSUInteger)ceilf((float)[layers count] / (float)
    itemsPerRow);
    //判断 rows 是否为空
    if (!rows) {
        rows = 1;
    }
    //布局
}

```

```

for (Layer *aLayer in layers) {
    NSUInteger indexInRow = index % itemsPerRow;
    NSUInteger row = index / itemsPerRow;
    aLayer.position = CGPointMake((indexInRow + 1) * (self.bounds.
        size.width / (itemsPerRow + 1)), (row + 1) * (self.bounds.
        size.height / (rows + 1)));    //设置位置
    if (![aLayer superlayer]) {
        [self.layer addSublayer:aLayer];    //添加图层对象
    }
    index++;
}
}

```

在触摸滚珠后，滚珠又会重新实现一种布局。需要使用 `timerMethod:`、`layoutSubviews`、`touchesBegan:withEvent:`、`touchesMoved:withEvent:`、`touchesEnded:withEvent:`、`touchesCancelled:withEvent:` 方法。其中，`touchesBegan:withEvent:`、`touchesMoved:withEvent:`、`touchesEnded:withEvent:`、`touchesCancelled:withEvent:` 方法实现触摸功能。程序代码如下：

```

//开始触摸
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    self.currentTouch = [[touches allObjects] lastObject]; //设置触摸点
    [self setNeedsLayout];
}
//移动触摸
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    self.currentTouch = [[touches allObjects] lastObject]; //设置触摸点
    [self setNeedsLayout];
}
//移动触摸
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    self.currentTouch = nil;    //将触摸点设置为空
    [self setNeedsLayout];
}
//取消触摸
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
    self.currentTouch = nil;    //将触摸点设置为空
    [self setNeedsLayout];
}

```

`timerMethod:` 方法是定时器方法，实现对圆半径的判断以及调用绘制方法。程序代码如下：

```

- (void)timerMethod:(NSTimer *)t
{
    //判断 animationFlag 是否为 YES
    if (animationFlag) {
        if ((radius += 5) > 100) {

```

```

        animationFlag = NO;                //设置 animationFlag
    }
}
else {
    if ((radius -= 5) < 60) {
        animationFlag = YES;                //设置 animationFlag
    }
}
if (++rotateIndex >= [layers count]) {
    rotateIndex = 0;
}
[self setNeedsLayout];                    //重新绘制
}

```

(8) 打开 ViewController.h 文件, 编写代码, 实现对头文件 View.h 的声明。

(9) 打开 ViewController.m 文件, 编写代码, 实现对视图 View 对象的创建以及添加。
程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    View *aView = [[View alloc] initWithFrame:[UIScreen mainScreen].bounds];
    //实例化对象
    aView.autoresizingMask = UIViewAutoresizingFlexibleHeight |
    UIViewAutoresizingFlexibleWidth; //自适应
    self.view = aView;
}

```

【代码解析】

本实例关键功能是圆珠以圆环形状排列以及动画效果。下面依次讲解这两个知识点。

1. 圆珠排列为圆环的效果

在本实例中当轻拍界面后, 原本以矩阵方式有序排列的滚珠会变为一个圆环排列, 在本实例中, 圆环的排列就会对滚珠的位置进行了重新的设置, 代码如下:

```

CGPoint p = [self.currentTouch locationInView:self];
NSUInteger index = rotateIndex;
for (Layer *aLayer in layers) {
    CGFloat r = M_PI * 2 * index / [layers count];
    CGFloat x = p.x + cos(r) * radius * -1;
    CGFloat y = p.y + sin(r) * radius * -1;
    aLayer.position = CGPointMake(x, y);
    index++;
}

```

2. 动画效果

在排列为圆环后, 圆环会缩放以及旋转, 它的实现使用了 timerMethod: 方法, 此方法会在每隔 0.1 秒后调用。

实例 54 永不消失的电波

【实例描述】

本实例实现的功能是一个永不消失的电波。此动画在运行后，会出现一个由小放大的圆（此圆即电波），并且到达指定大小后圆会慢慢地消失，从而出现新的电波动画。用户还可以通过调整滑块来改变圆的半径大小以及颜色。运行效果如图 4.12 所示。

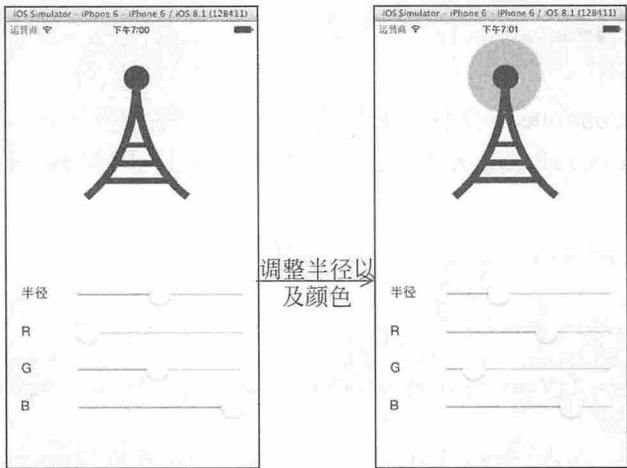


图 4.12 运行效果

【实现过程】

- (1) 创建一个项目，命名为“永不消失的电波”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 CALayer 类的 Layer 类。
- (4) 打开 Layer.h 文件，编写代码，实现属性的声明。程序代码如下：

```
#import <QuartzCore/QuartzCore.h>
@interface Layer : CALayer
//属性
@property (nonatomic, assign) CGFloat radius;
@property (nonatomic, assign) NSTimeInterval animationDuration;
@property (nonatomic, assign) NSTimeInterval pulseInterval;
@property (nonatomic, strong) CAAAnimationGroup *animationGroup;
@end
```

- (5) 打开 Layer.m 文件，编写代码，实现电波的动画效果。使用的方法如表 4-10 所示。

表 4-10 Layer.m 文件中方法总结

方 法	功 能
init	初始化
setRadius:	设置半径
setupAnimationGroup	创建动画效果

其中, init 方法实现对一些设置的初始化。程序代码如下:

```
- (id)init {
    self = [super init];
    if (self) {
        self.contentsScale = [UIScreen mainScreen].scale;
        self.opacity = 0; //设置透明度
        self.radius = 60; //设置半径
        self.animationDuration = 3; //设置动画持续时间
        self.pulseInterval = 0;
        //设置背景
        self.backgroundColor = [[UIColor colorWithRed:0.000 green:0.478
            blue:1.000 alpha:1] CGColor];
        dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_
            DEFAULT, 0), ^(void) {
            [self setupAnimationGroup];
            if(self.pulseInterval != INFINITY) {
                //通知主线程刷新
                dispatch_async(dispatch_get_main_queue(), ^(void) {
                    [self addAnimation:self.animationGroup forKey:@"pulse"];
                    //添加动画
                });
            }
        });
    }
    return self;
}
```

setRadius:方法实现对半径的设置。程序代码如下:

```
- (void)setRadius:(CGFloat)radius {
    _radius = radius;
    CGPoint tempPos = self.position;
    CGFloat diameter = self.radius * 2;
    self.bounds = CGRectMake(0, 0, diameter, diameter); //设置边界
    self.cornerRadius = self.radius; //设置半径
    self.position = tempPos; //设置位置
}
```

setupAnimationGroup 方法实现电波的动画效果。程序代码如下:

```
- (void)setupAnimationGroup {
    CAMediaTimingFunction *defaultCurve = [CAMediaTimingFunction function
        WithName:kCAMediaTimingFunctionDefault];
    //创建并设置 CAAAnimationGroup 对象,它允许多个动画同时播放
    self.animationGroup = [CAAnimationGroup animation];
    self.animationGroup.duration = self.animationDuration + self.
        pulseInterval; //设置动画持续时间
    self.animationGroup.repeatCount = INFINITY; //设置动画重复次数
    self.animationGroup.removedOnCompletion = NO;
    self.animationGroup.timingFunction = defaultCurve;
    //创建并设置 CABasicAnimation 对象,它是对单一动画的实现(即缩放动画的实现)
    CABasicAnimation *scaleAnimation = [CABasicAnimation animation
        WithKeyPath:@"transform.scale.xy"];
    scaleAnimation.fromValue = @0.0; //设置开始值
    scaleAnimation.toValue = @1.0; //设置结束值
}
```

```

scaleAnimation.duration = self.animationDuration; //设置动画持续时间
//创建并设置 CAKeyframeAnimation 对象，可以定义行动路线即渐渐消失的动画
CAKeyframeAnimation *opacityAnimation = [CAKeyframeAnimation animation
WithKeyPath:@"opacity"];
opacityAnimation.duration = self.animationDuration; //设置动画持续时间
opacityAnimation.values = @[@0.45, @0.45, @0]; //设置值
opacityAnimation.keyTimes = @[@0, @0.2, @1];
opacityAnimation.removedOnCompletion = NO;
NSArray *animations = @[scaleAnimation, opacityAnimation];
self.animationGroup.animations = animations //设置动画;
}

```

(6) 打开 ViewController.h 文件，编写代码，实现头文件、宏定义、对象、插座变量以及动作的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "Layer.h"
#define kMaxRadius 160
@interface ViewController : UIViewController{
    Layer *halo;
    //插座变量
    IBOutlet UIImageView *beaconView;
    IBOutlet UISlider *radiusSlider;
    IBOutlet UISlider *rSlider;
    IBOutlet UISlider *gSlider;
    IBOutlet UISlider *bSlider;
}
//动作
- (IBAction)radiusChanged:(id)sender;
- (IBAction)colorChanged:(id)sender;
@end

```

(7) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 4.13 所示。

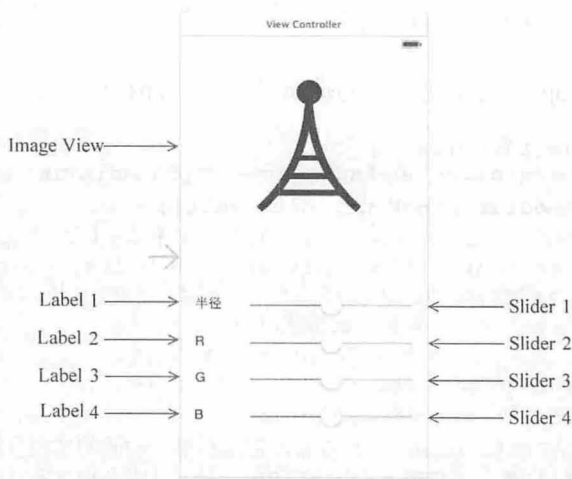


图 4.13 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 4-11 所示。

表 4-11 视图、控件设置

视图、控件	属性设置	其他
Image View1	Image: 1.png	
Label1	Text: 半径	
Label2	Text: R	
Label3	Text: G	
Label4	Text: B	
Slider1		与插座变量 radiusSlider 关联 与动作 radiusChanged:关联
Slider2		与插座变量 rSlider 关联 与动作 colorChanged:关联
Slider3		与插座变量 gSlider 关联 与动作 colorChanged:关联
Slider4		与插座变量 bSlider 关联 与动作 colorChanged:关联

(8) 打开 ViewController.m 文件, 编写代码, 实现对电波的创建、设置以及半径、颜色的改变。使用的方法如表 4-12 所示。

表 4-12 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
setupInitialValues	设置初始值
radiusChanged:	改变半径
colorChanged:	改变颜色

viewDidLoad 方法实现对电波图层的创建以及设置。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    halo = [Layer layer];
    halo.position = CGPointMake(beaconView.center.x+1, beaconView.center.y-70); //设置位置
    [self.view.layer insertSublayer:halo below:beaconView.layer];
    [self setupInitialValues];
}
```

setupInitialValues 方法实现对一些值的初始化设置。程序代码如下:

```
- (void)setupInitialValues {
    radiusSlider.value = 0.5; //设置值
    [self radiusChanged:nil];
    rSlider.value = 0;
    gSlider.value = 0.487; //设置 gSlider 滑块控件的值
    bSlider.value = 1.0;
    [self colorChanged:nil];
}
```

colorChanged:方法实现了对电波颜色的改变。程序代码如下:

```
- (IBAction)colorChanged:(id) sender {
```

```
UIColor *color = [UIColor colorWithRed:rSlider.value green:gSlider.
value blue:bSlider.value alpha:1.0];
halo.backgroundColor = color.CGColor;           //设置背景颜色
}
```

【代码解析】

本实例关键功能是电波的动画。下面就是这个知识点的详细讲解。在本实例中电波的动画效果是由两种动画共同实现的，具体步骤如下：

1. 电波的缩放动画

在本实例中，刚出现的电波是由 0 慢慢变为指定大小的，那么它的缩放动画是通过使用 CABasicAnimation 类实现的，代码如下：

```
CABasicAnimation *scaleAnimation = [CABasicAnimation animation WithKey
Path:@"transform.scale.xy"];
scaleAnimation.fromValue = @0.0;
scaleAnimation.toValue = @1.0;
scaleAnimation.duration = self.animationDuration;
```

2. 电波的消失

当电波到达指定位置后，会慢慢消失，它的消失动画是使用 CAKeyframeAnimation 关键帧动画来实现的，代码如下：

```
CAKeyframeAnimation *opacityAnimation = [CAKeyframeAnimation animation
WithKeyPath:@"opacity"];
opacityAnimation.duration = self.animationDuration;
opacityAnimation.values = @[@0.45, @0.45, @0];
opacityAnimation.keyTimes = @[@0, @0.2, @1];
opacityAnimation.removedOnCompletion = NO;
```

3. 组合动画

最后需要将电波的缩放动画和电波的消失动画同时进行播放，此时就需要使用到 CAAAnimationGroup 类来将这两种动画进行组合。代码如下：

```
self.animationGroup = [CAAnimationGroup animation];
self.animationGroup.duration = self.animationDuration + self.pulseInterval;
.....
NSArray *animations = @[opacityAnimation];
self.animationGroup.animations = animations;
```

实例 55 牛 顿 摆

【实例描述】

牛顿摆是一个于 19 世纪 60 年代被发明的桌面演示装置，5 个质量相同的球体由吊绳固定，彼此紧密排列。当摆动最右侧的球并在回摆时碰撞紧密排列的另外四个球时，最左边的球将被弹出，并且仅有最左边的球被弹出。当然此过程也是可逆的。本实例实现的功能就是这么一个牛顿摆，运行效果如图 4.14 所示。

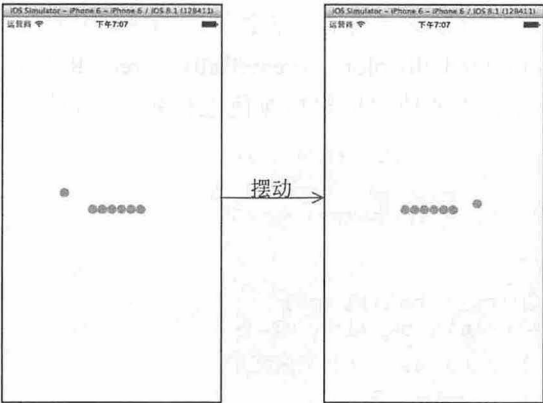


图 4.14 运行效果

【实现过程】

- (1) 创建一个项目，命名为“牛顿摆”。
- (2) 创建一个基于 UIView 类的 View 类。
- (3) 打开 View.h 文件，编写代码，实现属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface View : UIView
//属性
@property (nonatomic, strong) NSMutableArray *balls;
@property (nonatomic, strong) UIView *leftBall;
.....
@property (nonatomic, assign) BOOL shouldAnimate;
//方法
- (id)initWithFrame:(CGRect)frame;
- (id)initWithFrame:(CGRect)frame ballColor:(UIColor *)ballColor;
- (void)startAnimating; //开始动画
- (void)stopAnimating; //结束动画
- (BOOL)isAnimating;
@end
```

(4) 打开 View.m 文件，编写代码，实现对牛顿摆的功能，即实现对小球的绘制以及摆动功能。使用的方法如表 4-13 所示。

表 4-13 View.m文件中方法总结

方 法	功 能
initWithFrame:	使用框架初始化
initWithFrame:ballColor:	使用创建和球的颜色初始化
createBalls	创建球
createReflection	创建球的反射效果
ball	获取球
leftBallPendulate	左边球的摆动
rightBallPendulate	右边球的摆动
setAnchorPoint:forView:	设置固定的点
startAnimating	开始动画
stopAnimating	结束动画
isAnimating	获取是否需要动画

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。小球的绘制需要使用 `initWithFrame:`、`initWithFrame:ballColor:`、`createBalls`、`createReflection`、`ball` 方法。其中，`initWithFrame:ballColor:` 方法实现使用框架和颜色进行初始化设置。程序代码如下：

```
- (id)initWithFrame:(CGRect)frame ballColor:(UIColor *)ballColor
{
    self = [super initWithFrame:frame];
    if (self)
    {
        self.ballColor = ballColor; //设置球的颜色
        self.offset = sin(ballPendulateAngle) * (ballPendulateRadiusFactor
            + 0.5) * ballSize; //设置偏移量

        [self createBalls]; //创建球
        [self createReflection];
        self.shouldAnimate = YES;
        [self startAnimating]; //开始动画
    }
    return self;
}
```

`createBalls` 方法实现对牛顿摆中的小球的创建。程序代码如下：

```
- (void)createBalls
{
    self.balls = [NSMutableArray array]; //创建数组
    CGFloat width = CGRectGetWidth(self.frame);
    CGFloat xPos = width / 2 - (ballCount / 2 + 0.5) * ballSize;
    CGFloat yPos = CGRectGetHeight(self.frame) / 2 - ballSize / 2;
    //创建小球
    for (int i = 0; i < ballCount; i++)
    {
        UIView *ball = [self ball];
        ball.frame = CGRectMake(xPos, yPos, ballSize, ballSize); //设置框架
        [self addSubview:ball];
        [self.balls addObject:ball]; //添加对象
        xPos += ballSize;
    }
    self.leftBall = self.balls[0];
    self.rigthBall = self.balls[ballCount - 1];
}
```

`createReflection` 方法实现绘制小球反射效果（即倒影），这个方法主要是为了添加小球的美观性，读者可以选择性使用。程序代码如下：

```
- (void)createReflection
{
    self.reflectionBalls = [NSMutableArray array]; //创建数组
    CGFloat width = CGRectGetWidth(self.frame); //获取宽度
    CGFloat xPos = width / 2 - (ballCount / 2 + 0.5) * ballSize;
    CGFloat yPos = CGRectGetHeight(self.frame) / 2 + ballSize / 2 + 5;
    //循环，添加为数组添加对象以及实现视图对象的添加
    for (int i = 0; i < ballCount; i++)
    {
        UIView *reflectionBall = [self ball];
        reflectionBall.frame = CGRectMake(xPos, yPos, ballSize, ballSize);
    }
}
```

```

        //设置框架
        reflectionBall.transform = CGAffineTransformMakeRotation(M_PI);
        CAGradientLayer *gradient = [CAGradientLayer layer];
        gradient.frame = reflectionBall.bounds; //设置框架
        gradient.startPoint = CGPointMake(0.5, 1); //设置开始值
        gradient.endPoint = CGPointMake(0.5, 0);
        gradient.colors = @[ (id)[UIColor colorWithWhite:1 alpha:0.2].
        CGColor, (id)[UIColor clearColor].CGColor, (id)[UIColor clearColor].CGColor];
        gradient.locations = @[ (0), (0.35), (1)]; //设置位置
        reflectionBall.layer.mask = gradient;
        [self addSubview:reflectionBall]; //添加视图对象
        [self.reflectionBalls addObject:reflectionBall]; //添加对象
        xPos += ballSize;
    }
    self.leftReflectionBall = self.reflectionBalls[0];
    self.rightReflectionBall = self.reflectionBalls[ballCount - 1];
}

```

ball 方法实现获取小球的功能。程序代码如下:

```

- (UIView *)ball
{
    UIView *ball = [[UIView alloc] initWithFrame:CGRectMake(0, 0, ballSize,
    ballSize)];
    ball.backgroundColor = self.ballColor; //设置背景颜色
    ball.layer.cornerRadius = ballSize / 2; //设置圆角半径
    ball.clipsToBounds = YES;
    return ball;
}

```

牛顿摆的摆动需要使用 leftBallPendulate、rightBallPendulate、setAnchorPoint:forView:、startAnimating、stopAnimating、isAnimating 方法。其中, leftBallPendulate 方法实现对最左边小球的摆动的设置。程序代码如下:

```

- (void)leftBallPendulate
{
    [self setAnchorPoint:CGPointMake(0.5, -ballPendulateRadiusFactor)
    forView: self.leftBall];
    //最左边球的摆动效果
    [UIView animateWithDuration:0.2
    delay:0
    options:UIViewAnimationOptionCurveEaseOut
    animations:^(
        self.leftBall.transform = CGAffineTransformMakeRotation
        (ballPendulateAngle); //设置改变
        self.leftReflectionBall.frame = CGRectMake(self.
        leftReflectionBall.
        frame.origin.x - self.offset, self.leftReflection
        Ball.frame.origin.y, self.leftReflectionBall.frame.
        size.width, self.leftReflectionBall.frame.size.
        height); //设置框架
    )
    completion:^(BOOL finished) {
        [UIView animateWithDuration:0.2
        delay:0
        options:UIViewAnimationOptionCurveEaseIn

```

```

        animations:^(
            self.leftBall.transform = CGAffineTransform MakeRotation(0);
            //设置改变
            self.leftReflectionBall.frame =
                CGRectMake(self.leftReflection
                    Ball.frame.origin.x + self.offset,
                    self.leftReflectionBall.frame.
                        origin.y, self.leftReflection Ball.
                            frame.size.width, self.left
                                Reflection Ball.frame.size.height);
            //设置框架
        } completion:^(BOOL finished) {
            if (!_shouldAnimate)
            {
                [self rightBallPendulate];
            }
        }
    ];
}
}

```

setAnchorPoint: forView:方法实现对小球新点和旧点的设置。程序代码如下:

```

- (void) setAnchorPoint: (CGPoint) anchorPoint forView: (UIView *) view
{
    CGPoint newPoint = CGPointMake(view.bounds.size.width * anchorPoint.x,
        view.bounds.size.height * anchorPoint.y); //创建新的点
    CGPoint oldPoint = CGPointMake(view.bounds.size.width * view.layer.
        anchorPoint.x, view.bounds.size.height * view.layer.anchorPoint.y);
    //创建旧的点
    newPoint = CGPointApplyAffineTransform(newPoint, view.transform);
    oldPoint = CGPointApplyAffineTransform(oldPoint, view.transform);
    CGPoint position = view.layer.position; //获取位置
    position.x -= oldPoint.x;
    position.x += newPoint.x;
    position.y -= oldPoint.y;
    position.y += newPoint.y;
    view.layer.position = position; //设置位置
    view.layer.anchorPoint = anchorPoint;
}

```

(5) 打开 ViewController.h 文件, 编写代码, 实现头文件、对象的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "View.h"
@interface ViewController : UIViewController{
    View *ball;
}
@end

```

(6) 打开 ViewController.m 文件, 编写代码, 实现对具有牛顿摆视图的创建以及设置。程序代码如下:

```

- (void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    UIColor *ballColor = [UIColor colorWithRed:0.47 green:0.60 blue:0.89

```

```

    alpha:1];
    ballColor = [[View alloc] initWithFrame:self.view.bounds ballColor:
    ballColor]; //实例化对象
    [self.view addSubview:ballColor];
}

```

【代码解析】

本实例关键功能是牛顿摆中小球的摆动动画。下面就是这个知识点的详细讲解。

在本实例中小球的摆动动画需要使用 `animateWithDuration:delay:options:animations:completion:` 方法, 它可以实现动画的改变, 代码如下:

```

//动画效果
[UIView animateWithDuration:0.2
    delay:0
    options:UIViewAnimationOptionCurveEaseOut
    animations:^(
        //改变左边球的位置
        self.leftBall.transform = CGAffineTransformMakeRotation(ballPendulateAngle);
        self.leftReflectionBall.frame = CGRectMake(self.
        leftReflectionBall.frame.origin.x - self.offset,
        self.leftReflectionBall.frame.origin.y, self.left
        ReflectionBall.frame.size.width, self.left
        ReflectionBall.frame.size.height);
    ) completion:^(BOOL finished) {
        [UIView animateWithDuration:0.2
            delay:0
            options:UIViewAnimationOptionCurveEaseIn
            animations:^(
                //还原左边球的位置
                self.leftBall.transform = CGAffineTransformMakeRotation(0);
                self.leftReflectionBall.frame = CGRectMake(self.leftReflection
                Ball.frame.origin.x + self.offset,
                self.leftReflectionBall.frame.
                origin.y, self.leftReflection
                Ball.frame.size.width, self.left
                ReflectionBall.frame.size.height);
            ) completion:^(BOOL finished) {
                if (_shouldAnimate)
                {
                    [self rightBallPendulate];
                }
            }
        ];
    }
];

```

实例 56 摇 骰 子

【实例描述】

本实例实现的功能是一个简易的模拟摇骰子的游戏。当用户单击“摇骰子”按钮后, 碗中的骰子就会滚动起来, 经过一段时间后停止。运行效果如图 4.15 所示。

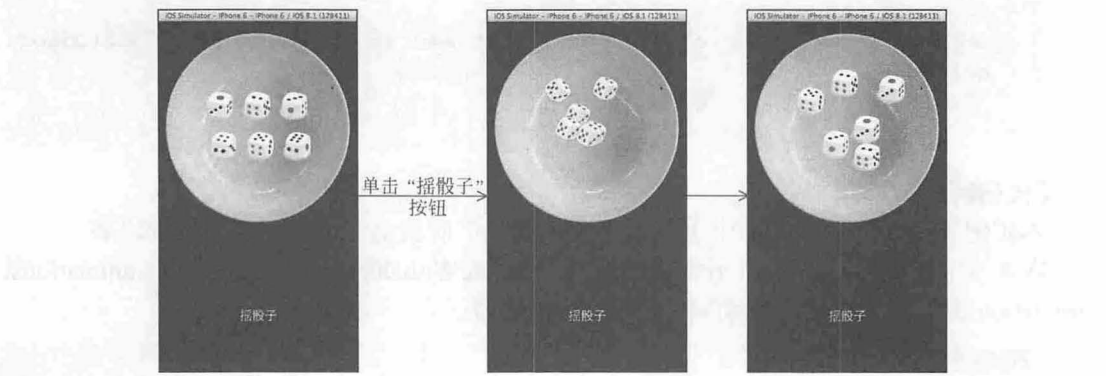


图 4.15 运行效果

【实现过程】

- (1) 创建一个项目，命名为“摇骰子”。
- (2) 添加图像 1.png~10.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现对象、动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //对象
    UIImageView *image1;
    UIImageView *image2;
    UIImageView *image3;
    .....
    UIImageView *dong5;
    UIImageView *dong6;
}
- (IBAction)yao:(id)sender;
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 4.16 所示。

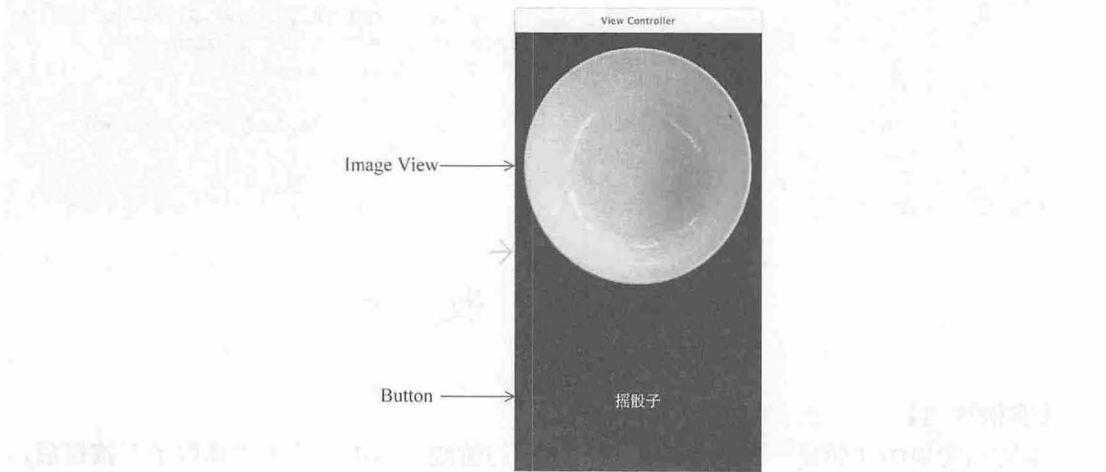


图 4.16 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 4-14 所示。

表 4-14 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 黑色	
Image View	Image: 10.png	
Button	Title: 摇骰子 Font: System 27.0 Text Color: 白色	与动作 yao:关联

(5) 打开 ViewController.m 文件, 编写代码, 实现摇骰子的功能。使用的方法如表 4-15 所示。

表 4-15 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
yao:	实现单击“摇骰子”按钮
animationDidStop:finished:	当动画结束时调用, 显示结果

其中, viewDidLoad 方法实现对视图对象的创建以及设置, 此图像视图作为骰子使用。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    UIImageView *_image1 = [[UIImageView alloc] initWithImage:[UIImage
    imageNamed:@"1.png"]];
    _image1.frame = CGRectMake(75.0, 115.0, 45.0, 45.0);          //设置框架
    _image1 = _image1;
    [self.view addSubview:_image1];                               //添加视图对象
    UIImageView *_image2 = [[UIImageView alloc] initWithImage:[UIImage
    imageNamed:@"2.png"]];
    _image2.frame = CGRectMake(135.0, 115.0, 45.0, 45.0);        //设置框架
    _image2 = _image2;
    [self.view addSubview:_image2];
    .....
    UIImageView *_image6 = [[UIImageView alloc] initWithImage:[UIImage
    imageNamed:@"6.png"]];
    _image6.frame = CGRectMake(200.0, 180.0, 45.0, 45.0);        //设置框架
    _image6 = _image6;
    [self.view addSubview:_image6];
}

```

yao:方法实现单击“摇骰子”按钮后实现的骰子的动画效果。首先, 需要将创建的图像视图隐藏, 即将骰子隐藏。程序代码如下:

```

image1.hidden = YES;
image2.hidden = YES;
image3.hidden = YES;
image4.hidden = YES;
.....
dong5.hidden = YES;

```

```
dong6.hidden = YES;
```

接着实现骰子旋转动画的初始化，程序代码如下：

```
//转动骰子的载入
NSArray *myImages = [NSArray arrayWithObjects:
    [UIImage imageNamed:@"7.png"],
    [UIImage imageNamed:@"8.png"],
    [UIImage imageNamed:@"9.png"], nil];

//骰子 1 的转动图片切换
UIImageView *dong11 = [[UIImageView alloc] initWithFrame:CGRectMake(85.0,
115.0, 45.0, 45.0)];
dong11.animationImages = myImages; //设置动画的图像
dong11.animationDuration = 0.5;
[dong11 startAnimating]; //开始动画
[self.view addSubview:dong11];
dong1 = dong11;
//骰子 2 的转动图片切换
UIImageView *dong12 = [[UIImageView alloc] initWithFrame:CGRectMake(135.0,
115.0, 45.0, 45.0)];
dong12.animationImages = myImages;
dong12.animationDuration = 0.5; //设置动画的持续时间
[dong12 startAnimating]; //开始动画
[self.view addSubview:dong12];
dong2 = dong12;
.....
//骰子 6 的转动图片切换
UIImageView *dong16 = [[UIImageView alloc] initWithFrame:CGRectMake(200.0,
180.0, 45.0, 45.0)];
dong16.animationImages = myImages; //设置动画的图像
dong16.animationDuration = 0.5; //设置动画的持续时间
[dong16 startAnimating];
[self.view addSubview:dong16];
dong6 = dong16;
```

其次实现动画的旋转。程序代码如下：

```
CABasicAnimation *spin = [CABasicAnimation animationWithKeyPath: @"trans
form.rotation"];
[spin setValue:[NSNumber numberWithFloat:M_PI * 16.0]]; //设置值
[spin setDuration:4]; //设置时间
```

然后，实现骰子的位置改变。程序代码如下：

```
//骰子 1 的位置变化
CGPoint p1 = CGPointMake(85.0, 115.0);
CGPoint p2 = CGPointMake(165.0, 100.0);
CGPoint p3 = CGPointMake(240.0, 160.0);
CGPoint p4 = CGPointMake(140.0, 200.0);
NSArray *keypoint = [[NSArray alloc] initWithObjects:[NSValue value WithCG
Point:p1],[NSValue valueWithCGPoint:p2],[NSValue valueWithCGPoint:p3],
[NSValue valueWithCGPoint:p4], nil]; //创建数组
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWith
KeyPath:@"position"];
[animation setValues:keypoint]; //设置值
[animation setDuration:4.0];
[animation setDelegate:self]; //设置委托
```

```

[dong11.layer setPosition:CGPointMake(140.0, 200.0)];
.....
//骰子 6 的位置变化
CGPoint p61 = CGPointMake(200.0, 180.0);
CGPoint p62 = CGPointMake(90.0, 190.0);
CGPoint p63 = CGPointMake(70.0, 140.0);
CGPoint p64 = CGPointMake(230.0, 110.0);
NSArray *keypoint6 = [[NSArray alloc] initWithObjects:[NSValue valueWithCGPoint:p61],[NSValue valueWithCGPoint:p62],[NSValue valueWithCGPoint:p63],
[NSValue valueWithCGPoint:p64], nil];
CAKeyframeAnimation *animation6 = [CAKeyframeAnimation animationWithKeyPath:@"position"];
[animation6 setValues:keypoint6];
[animation6 setDuration:4.0]; //设置时间
[animation6 setDelegate:self];
[dong16.layer setPosition:CGPointMake(230.0, 110.0)]; //设置位置

```

最后将这 6 个骰子的动画进行组合, 代码如下:

```

//骰子 1 的动画组合
CAAnimationGroup *animGroup = [CAAnimationGroup animation];
animGroup.animations = [NSArray arrayWithObjects: animation, spin, nil];
animGroup.duration = 4; //设置时间
[animGroup setDelegate:self];
[[dong11 layer] addAnimation:animGroup forKey:@"position"]; //添加动画
.....
//骰子 6 的动画组合
CAAnimationGroup *animGroup6 = [CAAnimationGroup animation];
animGroup6.animations = [NSArray arrayWithObjects: animation6, spin, nil];
animGroup6.duration = 4; //设置时间
[animGroup6 setDelegate:self];
[[dong16 layer] addAnimation:animGroup6 forKey:@"position"];
}

```

animationDidStop:finished:方法实现在动画结束后显示 6 个骰子的结果, 程序代码如下:

```

- (void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag
{
    //停止骰子自身的转动
    [dong1 stopAnimating];
    [dong2 stopAnimating];
    [dong3 stopAnimating];
    [dong4 stopAnimating];
    [dong5 stopAnimating];
    [dong6 stopAnimating];
    srand((unsigned)time(0));
    //骰子 1 的结果
    int result1 = (rand() % 5) + 1 ; //产生 1~6 的数
    switch (result1) {
        case 1:
            dong1.image = [UIImage imageNamed:@"1.png"]; //设置图像
            break;
        case 2:
            dong1.image = [UIImage imageNamed:@"2.png"]; //设置图像
            break;
        case 3:
            dong1.image = [UIImage imageNamed:@"3.png"]; //设置图像
            break;
    }
}

```

```

        case 4:
            dong1.image = [UIImage imageNamed:@"4.png"]; //设置图像
            break;
        case 5:
            dong1.image = [UIImage imageNamed:@"5.png"]; //设置图像
            break;
        case 6:
            dong1.image = [UIImage imageNamed:@"6.png"]; //设置图像
            break;
    }
    .....
    //骰子 6 的结果
    int result6 = (rand() % 5) + 1 ; //产生 1~6 的数
    switch (result6) {
        case 1:
            dong6.image = [UIImage imageNamed:@"1.png"]; //设置图像
            break;
        case 2:
            dong6.image = [UIImage imageNamed:@"2.png"]; //设置图像
            break;
        case 3:
            dong6.image = [UIImage imageNamed:@"3.png"]; //设置图像
            break;
        case 4:
            dong6.image = [UIImage imageNamed:@"4.png"]; //设置图像
            break;
        case 5:
            dong6.image = [UIImage imageNamed:@"5.png"]; //设置图像
            break;
        case 6:
            dong6.image = [UIImage imageNamed:@"6.png"]; //设置图像
            break;
    }
}

```

【代码解析】

本实例关键功能是骰子 3D 滚动的动画效果。下面就是这个知识点的详细讲解。

以其中的一个骰子为例，在本实例中，要想实现骰子 3D 滚动的动画效果，首先需要实现图像的切换，进而实现骰子转动的动画效果。代码如下：

```

NSArray *myImages = [NSArray arrayWithObjects:[UIImage imageNamed:@"7.png"],
[UIImage imageNamed:@"8.png"], [UIImage imageNamed:@"9.png"],nil];
UIImageView *dong11 = [[UIImageView alloc] initWithFrame:CGRectMake(85.0,
115.0, 45.0, 45.0)];
dong11.animationImages = myImages;
dong11.animationDuration = 0.5;
[dong11 startAnimating];
[self.view addSubview:dong11];
dong1 = dong11;

```

其次使用 CABasicAnimation 类实现骰子旋转的动画效果，代码如下：

```

CABasicAnimation *spin = [CABasicAnimation animationWithKeyPath: @"transform.rotation"];
[spin setValue:[NSNumber numberWithInt:M_PI * 16.0]];
[spin setDuration:4];

```

然后使用 `CAKeyframeAnimation` 关键帧动画实现骰子位置的改变, 代码如下:

```
CGPoint p1 = CGPointMake(85.0, 115.0);
CGPoint p2 = CGPointMake(165.0, 100.0);
CGPoint p3 = CGPointMake(240.0, 160.0);
CGPoint p4 = CGPointMake(140.0, 200.0);
NSArray *keypoint = [[NSArray alloc] initWithObjects:[NSValue valueWith
CGPoint:p1],[NSValue valueWithCGPoint:p2],[NSValue valueWithCGPoint: p3],
[NSValue valueWithCGPoint:p4], nil];
CAKeyframeAnimation *animation = [CAKeyframeAnimation animationWithKeyPath:
@"position"];
[animation setValues:keypoint];
[animation setDuration:4.0];
[animation setDelegate:self];
[dong11.layer setPosition:CGPointMake(140.0, 200.0)];
```

最后将旋转的动画效果和骰子位置改变的动画效果使用 `CAAnimationGroup` 类进行组合, 代码如下:

```
CAAnimationGroup *animGroup = [CAAnimationGroup animation];
animGroup.animations = [NSArray arrayWithObjects: animation, nil];
animGroup.duration = 4;
[animGroup setDelegate:self];
[[dong11 layer] addAnimation:animGroup forKey:@"position"];
```

实例 57 计 数 器

【实例描述】

本实例实现的功能就一个计数器的功能。在每个 1 秒后, 计数器中的内容会添加 1, 直到 9999 为止。运行效果如图 4.17 所示。

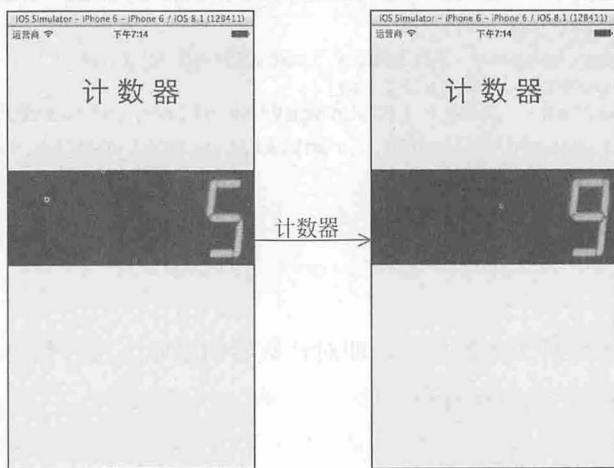


图 4.17 运行效果

【实现过程】

(1) 创建一个项目, 命名为“计数器”。

- (2) 添加图像 1.png、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 View 类。
- (4) 打开 View.h 文件，编写代码，实现属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface View : UIView
//属性
@property(nonatomic,assign)int length;
@property(nonatomic,assign)BOOL isInit;
@property(nonatomic,assign)int number;
-(void)showText:(int)number;           //显示文本
@end
```

- (5) 打开 View.m 文件，编写代码，实现计数器的功能。使用的方法如表 4-16 所示。

表 4-16 View.m文件中方法总结

方 法	功 能
beginLode:	添加图像
showText:	显示文本
lodeImageFoNumber:	获取图像

beginLode:方法实现对图像的添加，从而实现计数器的初始化。程序代码如下：

```
-(void)beginLode:(int)length
{
    //遍历，移除视图
    for (UIView* view in [self subviews]) {
        [view removeFromSuperview];
    }
    CGRect rect = self.frame;
    //遍历，添加视图对象
    for (int i= 0; i< length ; i++) {
        UIImageView* backImag = [[UIImageView alloc]initWith Frame:CG
            RectMake(i * rect.size.width/_length,0,rect.size. width/length,
            rect.size.height)];
        backImag.image = [UIImage imageNamed:@"1.png"];           //设置图像
        [self addSubview:backImag];
        UIImageView* Imag = [[UIImageView alloc]initWithFrame:CGRectMake(i
            * rect.size.width/length,0,rect.size.width/_length,rect.size.height)];
        Imag.tag = length - i;           //设置 tag 值
        [self addSubview:Imag];
    }
    self.backgroundColor = [UIColor clearColor];
}
```

showText:方法实现对文本的显示，即对计数器的显示。程序代码如下：

```
-(void)showText:(int)number
{
    self.number = number;
    int leng = self.length == 0?1:self.length;
    //判断 isInit 是否为 NO
    if (self.isInit == NO) {
        self.isInit = YES;
        [self beginLode:leng];
    }
}
```

```

    }
    //遍历,并隐藏指定的图像视图
    for (int i = 1; i <= self.length ; i++) {
        UIImageView* image = (UIImageView*)[self viewWithTag:i];
        image.hidden = YES;
    }
    for (int i = 1; i <= leng ; i++) {
        //判断电子表中的第一位
        if (i == 1) {
            int _right = number%10;
            UIImageView* image = (UIImageView*) [self viewWithTag:i];
            image.hidden = NO; //隐藏图像
            image.image = [self lodeImageFoNumber:_right]; //设置图像
        }else{
            long long int JieCheng = 10;
            for (int j = 1 ; j < i - 1 ; j++) {
                JieCheng = JieCheng *10;
            }
            long long int _life = number/JieCheng % 10;
            UIImageView* image = (UIImageView*) [self viewWithTag:i];
            //实例化对象
            //判断并显示图像
            if ( i<=[NSString stringWithFormat:@"%d",number] length)) {
                image.hidden = NO; //隐藏图像
            }
            image.image = [self lodeImageFoNumber:_life]; //设置图层
        }
    }
}

```

lodeImageFoNumber:方法实现在一张图像中获取指定区域的图像。程序代码如下:

```

-(UIImageView*)lodeImageFoNumber:(long long int)number
{
    CGRect imageRect = CGRectMake(number*36,0,36,48);
    UIImage* image = [UIImage imageNamed:@"2.png"];
    CGImageRef imageRef =CGImageCreateWithImageInRect(image.CGImage, imageRect);
    return [[UIImage alloc] initWithCGImage:imageRef]; //返回图像
}

```

(6) 打开 ViewController.h 文件,编写代码,实现头文件、对象、实例遍历的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "View.h"
@interface ViewController : UIViewController{
    IBOutlet View *vl; //声明插座变量
    NSTimer *timer; //声明时间定时器对象
    int i;
}
@end

```

(7) 打开 Main.storyboard 文件,对 View Controller 视图控制器的设计界面进行设计,效果如图 4.18 所示。

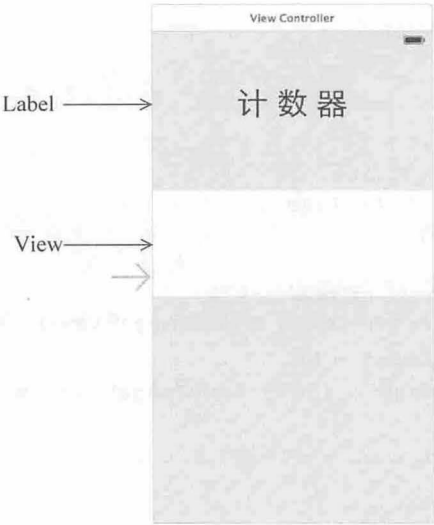


图 4.18 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 4-17 所示。

表 4-17 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	
Label	Text: 计数器 Font: System 36.0 Alignment: 居中	
View		Class: View 与插座变量 vl 关联

(8) 打开 ViewController.m 文件，编写代码，实现计数器的动画效果。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    i=9999;
    timer=[NSTimer scheduledTimerWithTimeInterval:1.0 target:self selector:
    @selector(show) userInfo:nil repeats:YES]; //创建时间定时器对象
    vl.length =4;
    [vl showText:1]; //调用 showText:方法，显示文本
}
-(void) show{
    i--;
    [vl showText:vl.number + 1]; //调用 showText:方法，显示文本
    //判断 i 是否为 0
    if (i==0) {
        [timer invalidate]; //让定时器失效
    }
}
```

【代码解析】

本实例关键功能是图像的截取以及数字的显示。下面依次讲解这两个知识点。

1. 图像的截取

在本实例中,图像的截取使用了 `CGImage` 的 `CGImageCreateWithImageInRect` 函数实现,它的功能是在一个给定的区域内创建位图。其语法形式如下:

```
CGImageRef CGImageCreateWithImageInRect (
    CGImageRef image,
    CGRect rect
);
```

其中, `CGImageRef image` 表示原始的图像; `CGRect rect` 表示给定的矩形区域。在本实例中的代码如下:

```
CGImageRef imageRef =CGImageCreateWithImageInRect (image.CGImage, imageRect);
```

2. 数字的显示

在本实例中,电子表中数字的显示就是调用了 `lodeImageFoNumber:` 方法,其中, `lodeImageFoNumber:` 方法正是实现了图像的裁剪功能。代码如下:

```
image.image = [self lodeImageFoNumber:_right];
```

实例 58 网格动画

【实例描述】

本实例功能是在图像中实现网格的动画效果。在每隔 6 秒后,会自动从左向右进行一次网格动画的播放。运行效果如图 4.19 所示。

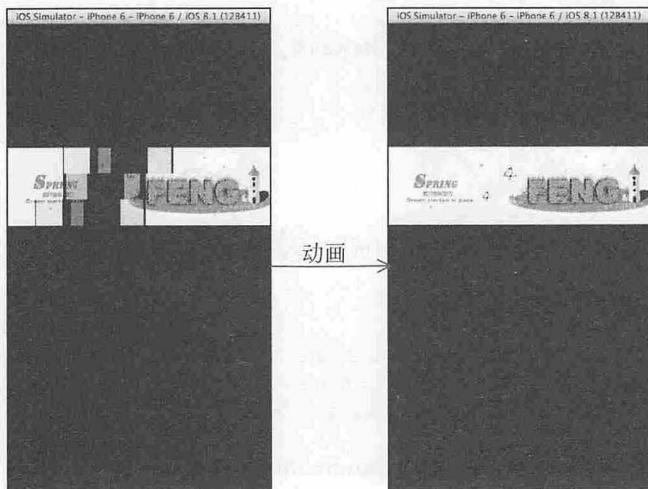


图 4.19 运行效果

【实现过程】

- (1) 创建一个项目，命名为“网格动画”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIViewController 类的 Grid 类。
- (4) 打开 Grid.h 文件，编写代码，实现属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface Grid : UIViewController
//属性
@property(strong) UIImage *image;
@property(strong) NSMutableArray *imageGrid;
@property(assign) CGSize gridCount;
//方法
-(id) initWithImage: (UIImage*) image gridCount: (CGSize) count;
-(void) doFlipAnimation;
@end
```

(5) 打开 Grid.m 文件，编写代码，实现网格的绘制以及网格动画的效果。使用的方法如表 4-18 所示。

表 4-18 Grid.m 文件中方法总结

方 法	功 能
initWithImage:gridCount:	初始化
generateImageGrid	生成网格
doFlipAnimation	翻转动画效果

网格绘制需要使用 initWithImage:gridCount: 和 generateImageGrid 方法。其中，initWithImage:gridCount: 方法使用初图像以及网格个数实现初始化功能，程序代码如下：

```
-(id) initWithImage: (UIImage*) image gridCount: (CGSize) count
{
    self.image = [image copy];
    self.gridCount = count; //设置网格个数
    if(self=[super init])
    {
        self.view.frame = CGRectMake(0, 0, image.size.width, image.size.height); //设置框架
        [self generateImageGrid]; //生成图像网格
    }
    return self;
}
```

generateImageGrid 方法实现网格的生成。程序代码如下：

```
-(void) generateImageGrid
{
    float gridSizeX = self.view.frame.size.width/self.gridCount.width;
    float gridSizeY = self.view.frame.size.height/self.gridCount.height;
    self.imageGrid = [[NSMutableArray alloc] init];
    //创建可变数组
    for(int yer=0; yer<self.gridCount.height; yer++)
    {
        for(int xer=0; xer<self.gridCount.width; xer++)
        {
```



```

//创建并设置图像视图
UIImageView *imgView = [[UIImageView alloc] initWithFrame: CGRectMake(
    xer*gridSizeX,yer*gridSizeY,gridSizeX,gridSizeY)];
imgView.contentMode = UIViewContentModeScaleAspectFill;
//设置图像的显示模式

//绘制网格图像
CGRect imgFrame = CGRectMake(xer*gridSizeX*2, yer*gridSizeY*
    [[UIScreen mainScreen] scale], gridSizeX*[[UIScreen mainScreen]
    scale], gridSizeY*2);
CGImageRef imageRef = CGImageCreateWithImageInRect([self.image
    CGImage], imgFrame);
UIImage* subImage = [UIImage imageWithCGImage: imageRef];
CGImageRelease(imageRef);
//添加网格图像到图像视图中
[imgView setImage: subImage];
[self.view addSubview:imgView];
[self.imageGrid addObject: imgView];
    }
}
[self doFlipAnimation];
}

```

doFlipAnimation 方法实现网格的快速翻转动画。程序代码如下:

```

-(void) doFlipAnimation
{
    //声明变量并赋值
    float delayDelta = 0.12f;
    float rowDelay = 0.06f;
    float duration = 0.5f;
    float maxDelayTime = 30.0f;
    float totalRowDelay = 0.0f;
    //循环
    for(int yer=0;yer<self.gridCount.height;yer++)
    {
        //循环
        for(int xer=0;xer<self.gridCount.width;xer++)
        {
            UIImageView *imgView = [self.imageGrid objectAtIndex: yer*self.
                gridCount.width+xer];
            float calculatedDelay = xer*delayDelta + totalRowDelay;
            //旋转动画
            CAKeyframeAnimation *rotationAnimation;
            rotationAnimation = [CAKeyframeAnimation animationWith KeyPath:
                @"transform.rotation.y"];
            rotationAnimation.keyTimes = [NSArray arrayWithObjects: [NSNum
                ber numberWithFloat: 0], [NSNumber numberWithFloat: 0.5], [NSN
                umber numberWithFloat: 0.50001], [NSNumber numberWithFloat: 1.0],
                nil];
            //设置关键时间
            rotationAnimation.values = [NSArray arrayWithObjects: [NSNumber
                numberWithFloat: 0], [NSNumber numberWithFloat: M_PI/2],
                [NSNumber numberWithFloat: -M_PI/2], [NSNumber numberWithFloat:
                0],nil];
            //设置值

```

```

rotationAnimation.duration = duration;           //设置动画持续时间
rotationAnimation.beginTime = CACurrentMediaTime()+calculatedDelay; //设置动画开始时间
rotationAnimation.fillMode = kCAFillModeForwards;
rotationAnimation.removedOnCompletion = NO;
rotationAnimation.timingFunction=[CAMediaTimingFunctionfunctionWithName:kCAMediaTimingFunctionLinear];
[imageView.layer addAnimation:rotationAnimation forKey:@"rotation Animation1"];           //添加动画
//改变位置的动画
CAKeyframeAnimation *opacityAnimation;
opacityAnimation = [CAKeyframeAnimation animationWith KeyPath:@"opacity"];
opacityAnimation.keyTimes = [NSArray arrayWithObjects: [NSNumber numberWithInt: 0], [NSNumber numberWithInt: 0.5], [NSNumber numberWithInt: 0.50001], [NSNumber numberWithInt: 1.0], nil];           //设置关键时间
opacityAnimation.values = [NSArray arrayWithObjects: [NSNumber numberWithInt: 1.0], [NSNumber numberWithInt: 0], [NSNumber numberWithInt: 0], [NSNumber numberWithInt: 1.0], nil];           //设置值
opacityAnimation.duration = duration;
opacityAnimation.beginTime = CACurrentMediaTime()+ calculatedDelay; //设置动画开始时间
opacityAnimation.timingFunction = [CAMediaTimingFunction functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
//设置动画开始和退出的快慢
[imageView.layer addAnimation:opacityAnimation forKey:@"opacity"];
}
totalRowDelay += rowDelay;
}
[NSTimer scheduledTimerWithTimeInterval:6.0f target:self selector:@selector(doFlipAnimation) userInfo:nil repeats:NO]; //创建定时器
}

```

(6) 打开 ViewController.h 文件，编写代码，实现头文件、属性的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "Grid.h"
@interface ViewController : UIViewController
@property(strong) Grid *flipGrid;
@end

```

(7) 打开 Main.storyboard 文件，将设计界面的背景设置为黑色。

(8) 打开 ViewController.m 文件，编写代码，实现创建和添加具有网格翻转动画的视图。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    self.flipGrid = [[Grid alloc] initWithImage: [UIImage imageNamed:@"1.png"] gridCount:CGSizeMake(20,6)]; //实例化对象
}

```

```

[self.view addSubview: self.flipGrid.view];
CGRect originalFrame = self.flipGrid.view.frame;           //获取框架
originalFrame.origin.y = 150;
self.flipGrid.view.frame = originalFrame;                 //设置框架
}

```

【代码解析】

本实例关键功能是网格的生成以及动画效果。下面就是这个知识点的详细讲解。

1. 网格的生成

在本实例中一个网格的生成使用了 `CGImageCreateWithImageInRect` 函数，但是对于多个网格，需要利用 `for` 循环，代码如下：

```

for(int yer=0;yer<self.gridCount.height;yer++)
{
    for(int xer=0;xer<self.gridCount.width;xer++)
    {
        UIImageView *imgView = [[UIImageView alloc] initWith Frame:
            CGRectMake(xer*gridSizeX,yer*gridSizeY,gridSizeX,gridSizeY)];
        imgView.contentMode = UIViewContentModeScaleAspectFill;
        CGRect imgFrame = CGRectMake(xer*gridSizeX*2, yer*gridSizeY*
            [[UIScreen mainScreen] scale], gridSizeX*[[UIScreen mainScreen]
            scale], gridSizeY*2);
        CGImageRef imageRef = CGImageCreateWithImageInRect([self.image
            CGImage], imgFrame);
        UIImage* subImage = [UIImage imageWithCGImage: imageRef];
        CGImageRelease(imageRef);
        [imgView setImage: subImage];
        [self.view addSubview:imgView];
        [self.imageGrid addObject: imgView];
    }
}

```

2. 动画效果

在本实例中快速翻转的动画效果使用了 `CAKeyframeAnimation` 关键帧动画，以旋转为例，代码如下：

```

CAKeyframeAnimation *rotationAnimation;
rotationAnimation = [CAKeyframeAnimation animationWithKey Path: @"transform.rotation.y"];
.....
[imgView.layer addAnimation:rotationAnimation forKey:@"rotation Animation1"];

```

实例 59 钟 表

【实例描述】

人们对时间的概念往往很强，专门用于显示时间的东西，有表、手机等。本实例就为

读者实现一个钟表的功能。运行效果如图 4.20 所示。

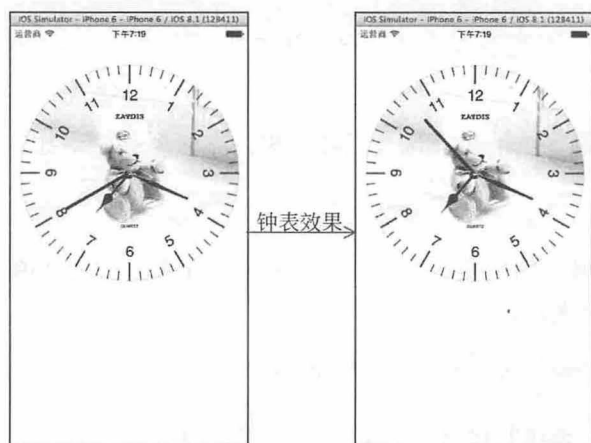


图 4.20 运行效果

【实现过程】

(1) 创建一个项目，命名为“钟表”。

(2) 添加图像 1.png、2.png、3.png、4.png 到创建项目的 Supporting Files 文件夹中。

(3) 创建一个基于 UIView 类的 ClockView 类。

(4) 打开 ClockView.h 文件，编写代码，实现宏定义、对象以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
//宏定义
#define HOURS_HAND_LENGTH 0.65
#define MIN_HAND_LENGTH 0.75
#define SEC_HAND_LENGTH 0.8
#define HOURS_HAND_WIDTH 10
#define MIN_HAND_WIDTH 8
#define SEC_HAND_WIDTH 4
@interface ClockView : UIView{
    //对象
    CALayer *containerLayer;
    CALayer *hourHand;
    CALayer *minHand;
    CALayer *secHand;
    NSTimer *timer;
}
//方法
- (void)start; //定时器的创建
- (void)stop;
- (void)setHourHandImage:(CGImageRef) image; //设置时针
- (void)setMinHandImage:(CGImageRef) image;
- (void)setSecHandImage:(CGImageRef) image;
- (void)setClockBackgroundImage:(CGImageRef) image; //设置钟表的背景
@end
```

(5) 打开 ClockView.m 文件, 编写代码, 实现钟表的功能。使用的方法如表 4-19 所示。

表 4-19 ClockView.m文件中方法总结

方 法	功 能
initWithFrame:	初始化表
start	定时器的创建
Stop	让定时器失效
setHourHandImage:	设置时针
setMinHandImage:	设置分针
setSecHandImage:	设置秒针
setClockBackgroundImage:	设置钟表的背景
Degrees2Radians	获取度数
updateClock:	更新钟表
layoutSubviews	布局

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, initWithFrame: 方法实现对钟表的初始化功能。程序代码如下:

```
- (id)initWithFrame:(CGRect) frame
{
    self = [super initWithFrame:frame];
    if (self) {
        //创建图层
        containerLayer = [CALayer layer] ;
        hourHand = [CALayer layer] ;
        minHand = [CALayer layer] ;
        secHand = [CALayer layer] ;
        //设置显示内容
        [self setClockBackgroundImage:NULL];
        [self setHourHandImage:NULL];
        [self setMinHandImage:NULL];
        [self setSecHandImage:NULL];
        //添加子图层
        [containerLayer addSublayer:hourHand];
        [containerLayer addSublayer:minHand];
        [containerLayer addSublayer:secHand];
        [self.layer addSublayer:containerLayer];
    }
    return self;
}
```

setHourHandImage:方法实现对钟表中的时针的设置。程序代码如下:

```
- (void)setHourHandImage:(CGImageRef) image
{
    //判断时针是否有图像
    if (image == NULL) {
        //如果没有图像
        hourHand.backgroundColor = [UIColor blackColor].CGColor;
        hourHand.cornerRadius = 3;
    }else{
        //如果有图像
        hourHand.backgroundColor = [UIColor clearColor].CGColor;
        hourHand.cornerRadius = 0.0;
    }
}
```



```

    hourHand.contents = (__bridge id)image;
}

```

updateClock:方法实现在每隔 1.0 秒后对时针的更新，即对时针、分针、秒针的更新。程序代码如下：

```

- (void) updateClock:(NSTimer *)theTimer{
    NSDateComponents *dateComponents = [[NSCalendar currentCalendar] components:(NSHourCalendarUnit | NSMinuteCalendarUnit | NSSecondCalendarUnit) fromDate:[NSDate date]];
    NSInteger seconds = [dateComponents second];           //获取秒
    NSInteger minutes = [dateComponents minute];           //获取分
    NSInteger hours = [dateComponents hour];                //获取小时
    //判断小时是否超过 12
    if (hours > 12)
        hours -=12;
    CGFloat secAngle = Degrees2Radians(seconds/60.0*360);
    CGFloat minAngle = Degrees2Radians(minutes/60.0*360);
    CGFloat hourAngle = Degrees2Radians(hours/12.0*360) + minAngle/12.0;
    //实现旋转
    secHand.transform = CATransform3DMakeRotation (secAngle+M_PI, 0, 0, 1);
    minHand.transform = CATransform3DMakeRotation (minAngle+M_PI, 0, 0, 1);
    hourHand.transform = CATransform3DMakeRotation (hourAngle+M_PI, 0, 0, 1);
}

```

layoutSubviews 方法实现的是对终点的布局功能。程序代码如下：

```

- (void) layoutSubviews
{
    [super layoutSubviews];
    containerLayer.frame = CGRectMake(0, 0, self.frame.size.width, self.frame.size.height);
    float length = MIN(self.frame.size.width, self.frame.size.height)/2; //旋转
    CGPoint c = CGPointMake(self.frame.size.width/2, self.frame.size.height/2);
    hourHand.position = minHand.position = secHand.position = c; //设置位置
    CGFloat w, h;
    //判断 hourHand 的 contents 属性值是否为 NULL
    if (hourHand.contents == NULL){
        w = HOURS_HAND_WIDTH;
        h = length*HOURS_HAND_LENGTH;
    }else{
        w = CGImageGetWidth((CGImageRef)hourHand.contents); //获取宽度
        h = CGImageGetHeight((CGImageRef)hourHand.contents); //获取高度
    }
    hourHand.bounds = CGRectMake(0,0,w,h); //设置边界
    //判断 hourHand 的 contents 属性值是否为 NULL
    if (minHand.contents == NULL){
        w = MIN_HAND_WIDTH;
        h = length*MIN_HAND_LENGTH;
    }else{
        w = CGImageGetWidth((CGImageRef)minHand.contents); //获取宽度
        h = CGImageGetHeight((CGImageRef)minHand.contents); //获取高度
    }
    minHand.bounds = CGRectMake(0,0,w,h);
    //判断 secHand 的 contents 属性值是否为 NULL
    if (secHand.contents == NULL){
        w = SEC_HAND_WIDTH;

```

```

        h = length*SEC_HAND_LENGTH;
    }else{
        w = CGImageGetWidth((CGImageRef) secHand.contents);    //获取宽度
        h = CGImageGetHeight((CGImageRef) secHand.contents);    //获取高度
    }
    secHand.bounds = CGRectMake(0,0,w,h);
    //设置锚点
    hourHand.anchorPoint = CGPointMake(0.5,0.0);
    minHand.anchorPoint = CGPointMake(0.5,0.0);
    secHand.anchorPoint = CGPointMake(0.5,0.0);
    containerLayer.anchorPoint = CGPointMake(0.5, 0.5);
}

```

(6) 打开 ViewController.h 文件, 编写代码, 实现对头文件、对象的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "ClockView.h"
@interface ViewController : UIViewController{
    ClockView *clockView;
}
@end

```

(7) 打开 ViewController.m 文件, 编写代码, 实现钟表对象的创建以及显示。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    clockView = [[ClockView alloc] initWithFrame:CGRectMake(10, 50, 300, 300)];
    [clockView setClockBackgroundImage:[UIImage imageNamed:@"4. png"].CGImage];    //设置背景颜色
    [clockView setHourHandImage:[UIImage imageNamed:@"1.png"].CGImage];
    [clockView setMinHandImage:[UIImage imageNamed:@"2.png"].CGImage];
    //设置分针的图像
    [clockView setSecHandImage:[UIImage imageNamed:@"3.png"].CGImage];
    clockView.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresizingFlexibleHeight;
    [self.view addSubview:clockView];    //添加图像
}

- (void)viewWillAppear:(BOOL)animated
{
    [clockView start];
}

```

【代码解析】

本实例关键功能是时分秒的获取以及动画效果。下面依次讲解这两个知识点。

1. 时分秒的获取

在本实例中, 要获取时分秒。首先, 需要创建一个 NSDateComponents 时间组件。代码如下:

```

NSDateComponents *dateComponents = [[NSCalendar currentCalendar] components:
(NSHourCalendarUnit | NSMinuteCalendarUnit | NSSecondCalendarUnit) from
Date:[NSDate date]];

```

然后，使用 `NSDateComponents` 的 `hour`、`minute`、`second` 方法获取时间的时分秒。以 `second` 方法为例，`second` 方法实现获取时间的秒数。其语法形式如下：

```
- (NSInteger) second;
```

其中，该方法的返回值类型为整型。在此实例中就是使用了 `second` 方法实现获取了当前时间的秒数，代码如下：

```
NSInteger seconds = [dateComponents second];
```

2. 动画效果

在本实例中，钟表是一个圆形的，此钟表划分为 12 大格，在每两格之间又分为了 5 小格。在每隔 1 秒后会将秒针旋转 1 小格，在每隔 60 秒后分针会旋转 1 小格，在每隔 3600 秒后时针会旋转 1 大格。这样的动画效果，首先需要创建一个时间定时器，代码如下：

```
timer = [NSTimer scheduledTimerWithTimeInterval:1.0 target:self selector:
@selector(updateClock:) userInfo:nil repeats:YES];
```

在每隔 1.0 秒后，调用 `updateClock:` 方法，其中，在此方法中对时分秒针的旋转度数进行了设置，代码如下：

```
CGFloat secAngle = Degrees2Radians(seconds/60.0*360);
CGFloat minAngle = Degrees2Radians(minutes/60.0*360);
CGFloat hourAngle = Degrees2Radians(hours/12.0*360) + minAngle/12.0;
secHand.transform = CATransform3DMakeRotation (secAngle+M_PI, 0, 0, 1);
minHand.transform = CATransform3DMakeRotation (minAngle+M_PI, 0, 0, 1);
hourHand.transform = CATransform3DMakeRotation (hourAngle+M_PI, 0, 0, 1);
```

实例 60 点赞的效果

【实例描述】

本实例实现的功能是实现一个类似于点赞的效果。当单击界面的按钮时，按钮的图像就会变为蓝色，并且在周围会出现一个爆炸的效果。用于显示文本内容的是标签控件。运行效果如图 4.21 所示。

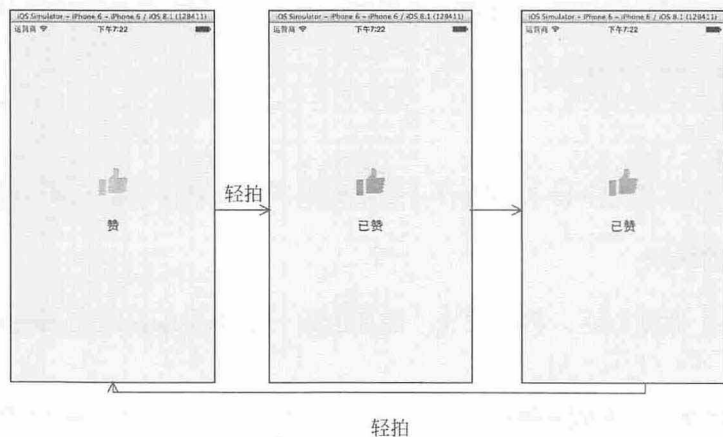


图 4.21 运行效果

【实现过程】

- (1) 创建一个项目，命名为“点赞的效果”。
- (2) 添加图像 1.png、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 View 类。
- (4) 打开 View.h 文件，编写代码，实现属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface View : UIView
//属性
@property (strong, nonatomic) UIImage *particleImage;
.....
@property (strong, nonatomic) CAEmitterLayer *explosionLayer;
- (void)animate;
@end
```

(5) 打开 View.m 文件，编写代码，实现在点赞时的爆炸效果。使用的方法如表 4-20 所示。

表 4-20 View.m文件中方法总结

方 法	功 能
setup	创建粒子系统以及粒子
initWithFrame:	使用框架实现初始化
initWithCoder:	初始化实例化的对象
layoutSubviews	布局
animate	动画
explode	爆炸效果
stop	停止动画
setParticleImage:	设置粒子的图像
setParticleScale:	设置粒子的缩放
setParticleScaleRange:	设置粒子缩放的范围

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，setup 方法实现对粒子对象以及粒子发射器对象的创建以及设置。程序代码如下：

```
- (void)setup {
    self.clipsToBounds = NO;
    self.userInteractionEnabled = NO;
    //创建并设置粒子对象
    CAEmitterCell *explosionCell = [CAEmitterCell emitterCell];
    explosionCell.name = @"explosion";
    explosionCell.alphaRange = 0.20;           //一个粒子的透明度能改变的范围
    explosionCell.alphaSpeed = -1.0;           //粒子透明度在生命周期内的改变速度
    explosionCell.lifetime = 0.7;               //生命周期范围
    explosionCell.lifetimeRange = 0.3;
    explosionCell.birthRate = 0;               //粒子参数的速度乘数因子
    explosionCell.velocity = 40.00;            //速度
    explosionCell.velocityRange = 10.00;       //速度范围
    //创建并设置粒子发射器对象
    _explosionLayer = [CAEmitterLayer layer];
    _explosionLayer.name = @"emitterLayer";
    _explosionLayer.emitterShape = kCAEmitterLayerCircle; //发射形状
```

```

_explosionLayer.emitterMode = kCAEmitterLayerOutline;
_explosionLayer.emitterSize = CGSizeMake(25, 0); //发射源的尺寸大小
_explosionLayer.emitterCells = @[explosionCell];
_explosionLayer.renderMode = kCAEmitterLayerOldestFirst; //渲染模式
_explosionLayer.masksToBounds = NO;
_explosionLayer.seed = 1366128504;
[self.layer addSublayer:_explosionLayer]; //添加图层对象
//创建并设置粒子对象
CAEmitterCell *chargeCell = [CAEmitterCell emitterCell];
chargeCell.name = @"charge";
chargeCell.alphaRange = 0.20; //一个粒子的透明度能改变的范围
chargeCell.alphaSpeed = -1.0; //粒子透明度在生命周期内的改变速度
chargeCell.lifetime = 0.3; //生命周期范围
chargeCell.lifetimeRange = 0.1;
chargeCell.birthRate = 0;
chargeCell.velocity = -40.0; //速度
chargeCell.velocityRange = 0.00;
//创建并设置粒子发射器对象
_chargeLayer = [CAEmitterLayer layer];
_chargeLayer.name = @"emitterLayer";
_chargeLayer.emitterShape = kCAEmitterLayerCircle; //发射形状
_chargeLayer.emitterMode = kCAEmitterLayerOutline;
_chargeLayer.emitterSize = CGSizeMake(25, 0); //发射源的尺寸大小
_chargeLayer.emitterCells = @[chargeCell];
_chargeLayer.renderMode = kCAEmitterLayerOldestFirst; //渲染模式
_chargeLayer.masksToBounds = NO;
_chargeLayer.seed = 1366128504;
[self.layer addSublayer:_chargeLayer]; //添加图层对象
self.emitterCells = @[chargeCell, explosionCell];
}

```

layoutSubviews 方法实现对粒子发射器的布局。程序代码如下：

```

- (void)layoutSubviews {
    [super layoutSubviews];
    CGPoint center = CGPointMake(CGRectGetMidX(self.bounds), CGRectGetMidY(self.bounds));
    self.chargeLayer.emitterPosition = center; //设置位置
    self.explosionLayer.emitterPosition = center;
}

```

animate 方法实现对爆炸效果的调用，从而实现动画效果。程序代码如下：

```

- (void)animate {
    self.chargeLayer.beginTime = CACurrentMediaTime(); //设置动画的开始时间
    [self.chargeLayer setValue:@80 forKeyPath:@"emitterCells.charge.birthRate"];
    [self performSelector:@selector(explode) withObject:nil afterDelay:0.2];
}

```

explode 方法实现爆炸效果。程序代码如下：

```

- (void)explode {
    [self.chargeLayer setValue:@0 forKeyPath:@"emitterCells.charge.birthRate"];
    self.explosionLayer.beginTime = CACurrentMediaTime(); //设置动画的开始时间
    [self.explosionLayer setValue:@500 forKeyPath:@"emitterCells.explosion.birthRate"];
    [self performSelector:@selector(stop) withObject:nil afterDelay:0.1];
}

```

(6) 创建一个基于 UIButton 类的 Button 类。

(7) 打开 Button.h 文件, 编写代码, 实现头文件、属性以及方法的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import "View.h"
@interface Button : UIButton
//属性
@property (strong, nonatomic) UIImage *particleImage;
@property (assign, nonatomic) CGFloat particleScale;
@property (assign, nonatomic) CGFloat particleScaleRange;
@property (strong, nonatomic) View *fireworksView;
//方法
- (void)animate;
- (void)popOutsideWithDuration:(NSTimeInterval)duration;
- (void)popInsideWithDuration:(NSTimeInterval)duration;
@end
```

(8) 打开 Button.m 文件, 编写代码, 实现点赞的效果。使用的方法如表 4-21 所示。

表 4-21 Button.m文件中方法总结

方 法	功 能
setup	创建并添加爆炸视图
initWithFrame:	初始化
initWithCoder:	初始化实例化的对象
layoutSubviews	布局
animate	爆炸的动画
popOutsideWithDuration:	按钮的放大
popInsideWithDuration:	按钮的缩小
particleImage	获取粒子图像
setParticleImage:	设置粒子图像
particleScale	获取缩放
setParticleScale:	设置缩放
particleScaleRange	获取缩放范围
setParticleScaleRange:	设置缩放范围

其中, setup 方法实现对爆炸视图的创建以及添加。程序代码如下:

```
- (void)setup {
    self.clipsToBounds = NO;
    _fireworksView = [[View alloc] init];
    [self addSubview:_fireworksView atIndex:0]; //插入视图对象
}
```

layoutSubviews 方法实现爆炸视图的布局。程序代码如下:

```
- (void)layoutSubviews {
    [super layoutSubviews];
    self.fireworksView.frame = self.bounds; //设置框架
    [self addSubview:self.fireworksView atIndex:0];
}
```

popOutsideWithDuration:方法实现按钮放大的动画效果。程序代码如下:

```
- (void)popOutsideWithDuration:(NSTimeInterval)duration {
    __weak typeof(self) weakSelf = self;
    self.transform = CGAffineTransformIdentity;
```

```

//为当前视图创建一个可以用于设置基本关键帧动画的 block 对象
[UIView animateKeyframesWithDuration:duration delay:0 options:0 animations: ^{
    //指定一个关键的单个帧的时间和动画
    [UIViewaddKeyframeWithRelativeStartTime:0relativeDuration:1/3.0animations:^(
        typeof(self) strongSelf = weakSelf;
        strongSelf.transform = CGAffineTransformMakeScale(1.5, 1.5); //缩放
    )];
    [UIView addKeyframeWithRelativeStartTime:1/3.0 relativeDuration:
    1/3.0 animations: ^{
        typeof(self) strongSelf = weakSelf;
        strongSelf.transform = CGAffineTransformMakeScale(0.8, 0.8); //缩放
    }];
    [UIView addKeyframeWithRelativeStartTime:2/3.0 relativeDuration:
    1/3.0 animations: ^{
        typeof(self) strongSelf = weakSelf;
        strongSelf.transform = CGAffineTransformMakeScale(1.0, 1.0); //缩放
    }];
} completion:nil];
}

```

(9) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量、实例变量以及动作的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "Button.h"
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet Button *button;
    BOOL _selected;
    IBOutlet UILabel *label;
}
- (IBAction)select:(id)sender;
@end

```

(10) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 4.22 所示。

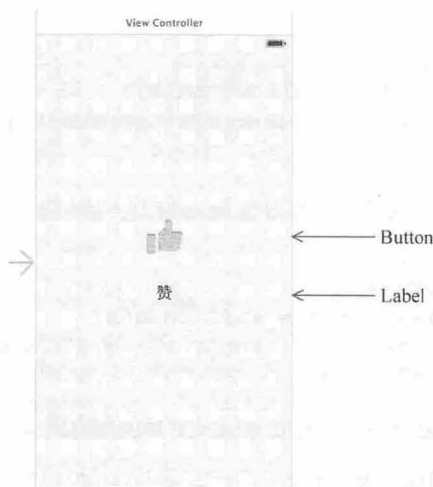


图 4.22 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 4-22 所示。

表 4-22 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	
Label	Text: 赞 Font: System 20.0 Alignment: 居中	与插座变量 label 关联
Button	Title: (空) Image: 2.png	Class: Button 与插座变量 button 关联 与动作 select:关联

(11) 打开 ViewController.m 文件, 编写代码, 实现对按钮的设置以及选择。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    button.imageView.image=[UIImage imageNamed:@"3.png"];    //设置图像
    button.imageView.scale = 0.05;                            //设置缩放值
    button.imageView.scaleRange = 0.02;                       //设置缩放范围
}

- (IBAction)select:(id)sender {
    _selected = !_selected;
    //判断按钮是否被选中
    if (_selected) {
        [button popOutsideWithDuration:0.5];
        [button setImage:[UIImage imageNamed:@"1"] forState:UIControlStateNormal];    //设置图像
        [button animate];
        label.text=@"已赞" ;
    }
    else {
        [button popInsideWithDuration:0.4];
        [button setImage:[UIImage imageNamed:@"2"] forState:UIControlStateNormal];    //设置图像
        label.text=@"赞";
    }
}

```

【代码解析】

由于本实例中的代码和方法非常多, 为了方便读者的阅读, 笔者绘制了一些执行流程图如图 4.23 和图 4.24 所示。其中, 界面的显示需要使用 initWithCoder:(Button.m)、setup(Button.m)、layoutSubviews(Button.m)、setImage(Button.m)、setScale(Button.m)、setScaleRange(Button.m)、viewDidLoad、initWithFrame:(View.m)、setup(View.m)、setImage(View.m)、setScale(View.m)、setScaleRange(View.m)、layoutSubviews(View.m)方法共同实现。它们的执行流程如图 4.23 所示。

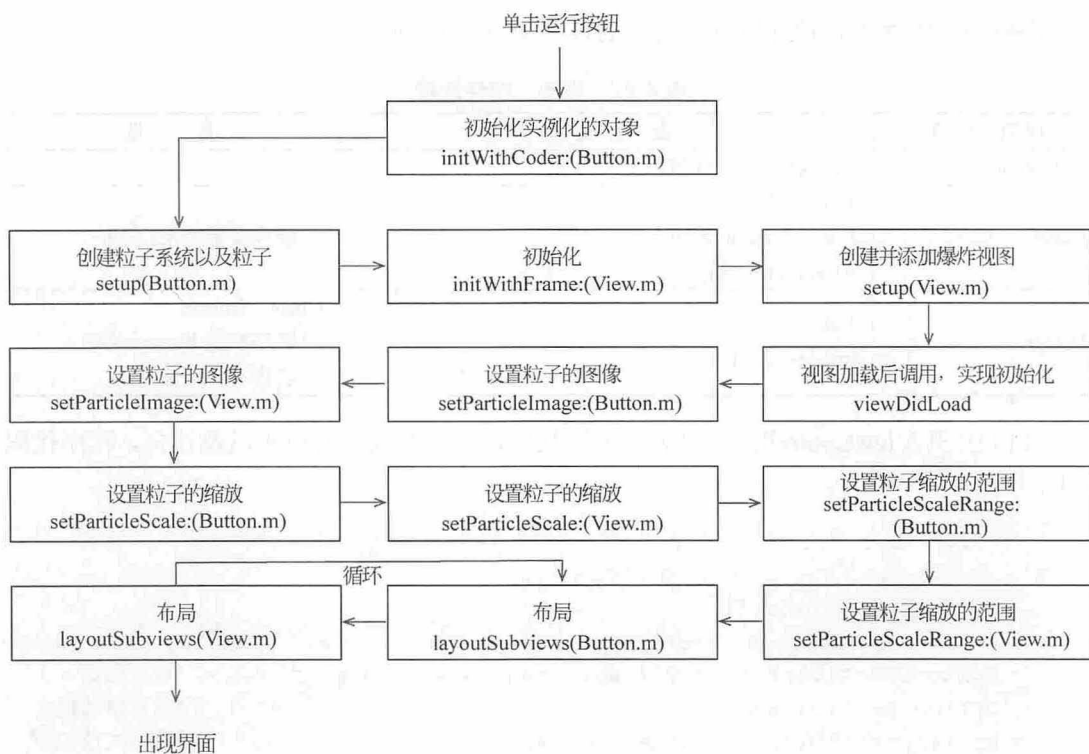


图 4.23 程序执行流程

如果想要实现点赞的效果，需要使用 `layoutSubviews(Button.m)`、`popOutsideWithDuration:`、`animate(Button.m)`、`layoutSubviews(Button.m)`、`animate(View.m)`、`explode`、`stop` 方法共同实现。它们的执行流程如图 4.24 所示。



图 4.24 程序执行流程

第5章 网页视图

在 iPhone 手机或者 iPad 中，人们通常使用 Safari 来进行网页的浏览，Safari 最主要的实现部分就是网页视图。在 iOS 开发中，也提供了网页视图组件。网页视图就相当于一个浏览器，人们可以使用它去浏览网址相应的内容，还可以加载自己的 HTML 内容。本章将主要讲解网页视图相关的实例。

实例 61 紧急求救中心

【实例描述】

本实例的功能是通过网页视图的识别电话号码功能，实现一个紧急求救中心的实例。当用户长按电话号码，就会出现一个动作表单，单击 Call *****就可以打电话了。运行效果如图 5.1 所示。

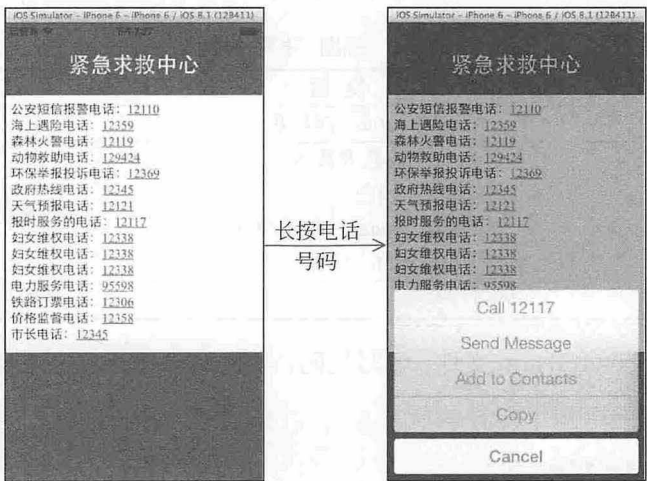


图 5.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“紧急求救中心”。
- (2) 打开 ViewController.h 文件，编写代码，实现插座变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    IBOutlet UIWebView *webView;
}
@end
```

//插座变量

(3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 5.2 所示。

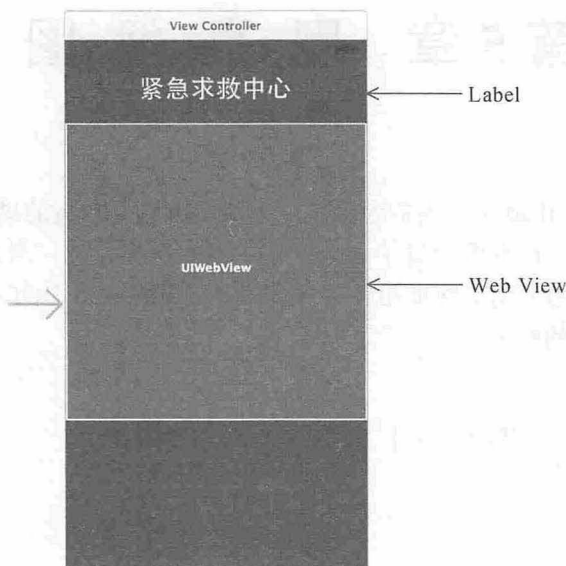


图 5.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-1 所示。

表 5-1 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 深红色	
Label	Text: 紧急求救中心 Color: 白色 Font: System Bold 27.0 Alignment: 居中	与插座变量 label 关联
Web View		与插座变量 webView 关联

(4) 打开 ViewController.m 文件，编写代码，实现 HTML 代码的加载。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    NSString *html1=@"公安短信报警电话: 12110<br>海上遇险电话: 12359<br>森林火
    警电话: 12119<br>动物救助电话: 129424 <br>环保举报投诉电话: 12369 <br>政府热
    线电话: 12345<br>天气预报电话: 12121 <br>报时服务的电话: 12117<br>妇女维权电
    话: 12338<br>妇女维权电话: 12338<br>妇女维权电话: 12338<br>电力服务电话:
    95598<br>铁路订票电话: 12306<br>价格监督电话: 12358<br>市长电话: 12345";
    [webView loadHTMLString:html1 baseURL:nil];           //加载 HTML 代码
}
```

【代码解析】

本实例关键功能是 HTML 代码的加载以及电话号码的识别。下面依次讲解这两个知

识点。

1. HTML代码的加载

HTML 代码的加载可以使用 `UIWebView` 的 `loadHTMLString:baseURL:` 方法实现, 其语法形式如下:

```
- (void)loadHTMLString:(NSString *)string baseURL:(NSURL *)baseURL;
```

其中, `(NSString *)string` 表示字符串对象, 用来设置页面的内容; `(NSURL *)baseURL` 表示内容的基 URL。在本实例中就使用了 `loadHTMLString:baseURL:` 方法来实现 html 代码的加载, 代码如下:

```
[webView loadHTMLString:html1 baseURL:nil];
```

其中, `html1` 表示字符串对象, 用来设置页面的内容; `nil` 表示没有内容的基 URL。

2. 电话号码的识别

在 `UIWebView` 中电话号码的识别是默认的, 可以在属性查看器中进行设置。也可以使用 `UIWebView` 的 `dataDetectorTypes` 属性对其进行设置, 其语法形式如下:

```
@property(nonatomic) UIDataDetectorTypes dataDetectorTypes;
```

`dataDetectorTypes` 属性除了可以识别电话号码外还可以识别网址、地址等信息, 如表 5-2 所示。

表 5-2 dataDetectorTypes属性值

属 性 值	功 能
<code>UIDataDetectorTypePhoneNumber</code>	识别电话号码
<code>UIDataDetectorTypeLink</code>	识别网址、链接
<code>UIDataDetectorTypeAddress</code>	识别地址
<code>UIDataDetectorTypeCalendarEvent</code>	识别时间
<code>UIDataDetectorTypeNone</code>	全都不识别
<code>UIDataDetectorTypeAll</code>	全部识别

实例 62 常用网址大全

【实例描述】

本实例实现的功能是一个常用网址大全的应用。当用户单击某一个导航, 就会打开相应的链接。运行效果如图 5.3 所示。

【实现过程】

- (1) 创建一个项目, 命名为“常用网址大全”。
- (2) 添加图像 1.jpg、2.png、3.png、4.png、5.png、6.png、7.png、8.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 `UIButton` 类的 `Button` 类。

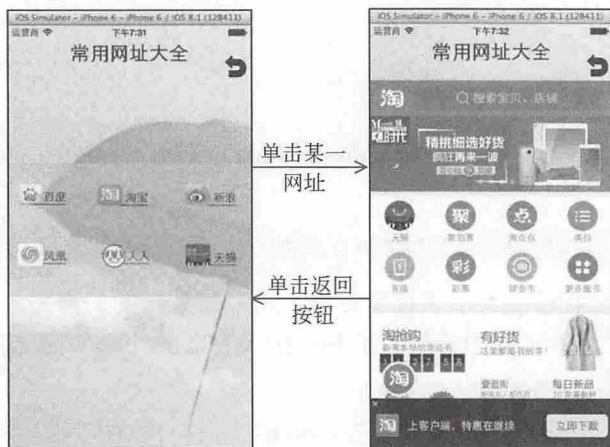


图 5.3 运行效果

(4) 打开 Button.m 文件，编写代码，实现对按钮下划线的绘制。程序代码如下：

```
(void) drawRect:(CGRect)rect {
    CGRect textRect = self.titleLabel.frame;
    CGFloat descender = self.titleLabel.font.descender;
    CGContextRef contextRef = UIGraphicsGetCurrentContext();
    CGContextSetStrokeColorWithColor(contextRef,
        self.titleLabel.textColor.CGColor); //设置线的颜色
    CGContextMoveToPoint(contextRef, textRect.origin.x, textRect.origin.y
        + textRect.size.height + descender+5); //设置开始点
    CGContextAddLineToPoint(contextRef, textRect.origin.x + textRect.size.
        width, textRect.origin.y + textRect.size.height + descender+5);
        //设置接收点
    CGContextClosePath(contextRef);
    CGContextDrawPath(contextRef, kCGPathStroke);
}
```

(5) 打开 ViewController.h 文件，编写代码，实现插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet UIWebView *webView ;
    IBOutlet UIView *vv;
}
//动作
- (IBAction)back:(id) sender;
- (IBAction)baidu:(id) sender; //单击按钮打开百度的网页
- (IBAction)taobao:(id) sender;
- (IBAction)xinlang:(id) sender; //单击按钮打开新浪的网页
- (IBAction)fenghuang:(id) sender;
- (IBAction)renren:(id) sender;
- (IBAction)tianmao:(id) sender;
@end
```

(6) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 5.4 所示。

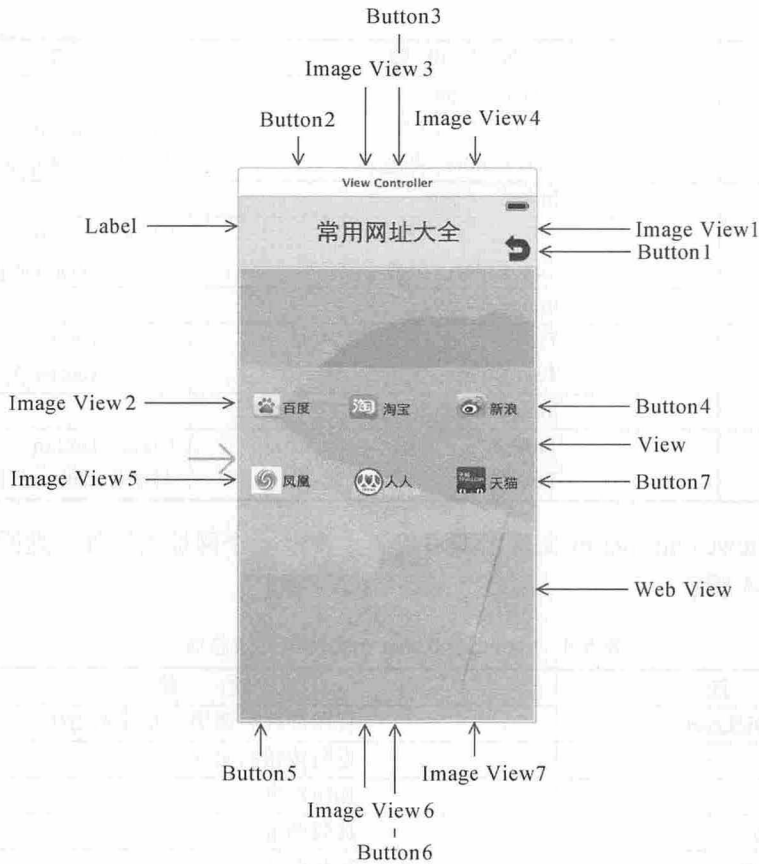


图 5.4 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-3 所示。

表 5-3 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View1	Image: 1.jpg	
Label	Text: 常用网址大全 Font: System 26.0 Alignment: 居中	
Button1	Title: (空) Background: 2.png	与动作 back:关联
View	Alpha: 0.75 Background: 浅灰色	与插座变量 vv 关联
Web View		与插座变量 webView 关联
Image View2	Image: 3.png	
Button2	Title: 百度 Text Color: 黑色	Class: Button 与动作 baidu:关联
Image View3	Image: 4.png	
Button3	Title: 淘宝 Text Color: 黑色	Class: Button 与动作 taobao:关联

续表

视图、控件	属 性 设 置	其 他
Image View4	Image: 5.png	
Button4	Title: 新浪 Text Color: 黑色	Class: Button 与动作 xinlang:关联
Image View5	Image: 6.jpg	
Button5	Title: 凤凰 Text Color: 黑色	Class: Button 与动作 fenghuang:关联
Image View6	Image: 7.png	
Button6	Title: 人人 Text Color: 黑色	Class: Button 与动作 renren:关联
Image View7	Image: 8.png	
Button7	Title: 天猫 Text Color: 黑色	Class: Button 与动作 tianmao:关联

(7) 打开 ViewController.m 文件, 编写代码, 实现 6 个网址的加载、返回等操作。使用的方法如表 5-4 所示。

表 5-4 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
back:	返回按钮的实现
baidu:	加载百度
taobao:	加载淘宝
xinlang:	加载新浪
fenghuang:	加载凤凰
renren:	加载人人
tianmao:	加载天猫

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, back:方法实现单击返回按钮后, 返回主界面的功能。程序代码如下:

```
- (IBAction)back:(id) sender {  
    webView.hidden=YES;  
    vv.hidden=NO;  
}
```

baidu:方法实现单击“百度”按钮后实现百度网址的加载。程序代码如下:

```
- (IBAction)baidu:(id) sender {  
    webView.hidden=NO;  
    vv.hidden=YES;  
    NSURL *url=[NSURL URLWithString:@"http://www.baidu.com"];  
    //创建 NSURLRequest 对象, 传递网址  
    NSURLRequest *request=[NSURLRequest requestWithURL:url];  
    //加载网址对应的网页内容  
    [webView loadRequest:request];  
}
```

【代码解析】

本实例关键功能是 URL 的加载。下面就是这个知识点的详细讲解。

网页视图加载 URL 网址，需要使用 UIWebView 的 loadRequest:方法，其语法形式如下：

```
- (void)loadRequest:(NSURLRequest *)request;
```

其中，(NSURLRequest *)request 表示 URL 请求标识。在本实例中就使用了 loadRequest:方法实现了网页视图加载 URL 网址，代码如下：

```
[webView loadRequest:request];
```

其中，request 表示 URL 请求标识。

实例 63 改变网页视图中字体的大小

【实例描述】

本实例实现的功能是改变网页视图中加载内容的字体大小。当用户滑动滑块时，字体大小就会发生相应的改变。运行效果如图 5.5 所示。



图 5.5 运行效果

【实现过程】

- (1) 创建一个项目，命名为“改变网页视图中字体的大小”。
- (2) 打开 ViewController.h 文件，编写代码，实现插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //声明插座变量
    IBOutlet UISlider *Slider;
    IBOutlet UIWebView *webView;
}
- (IBAction)change:(id) sender;
@end
```

- (3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，

效果如图 5.6 所示。

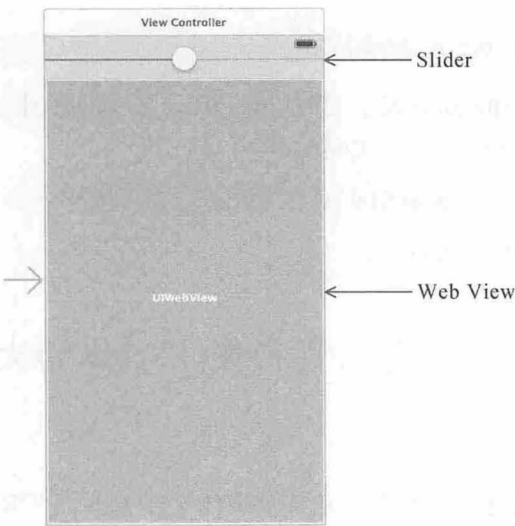


图 5.6 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-5 所示。

表 5-5 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	
Slider		与插座变量 Slider 关联 与动作 change:关联
Web View		与插座变量 webView 关联

(4) 打开 ViewController.m 文件，编写代码，实现在调整网页视图中字体的大小。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    Slider.maximumValue = 1000.0f;           //设置滑块控件的最大值
    Slider.minimumValue = 10.0f;              //设置滑块控件的最小值
    Slider.value = 10.0f;                     //设置当前的值
    NSURL* url = [NSURL URLWithString:@"http://www.baidu.com"];
    NSURLRequest* request = [[NSURLRequest alloc] initWithURL:url];
    [webView loadRequest:request];            //加载
}

- (IBAction)change:(id)sender {
    NSString* str1 = [NSString stringWithFormat:
@"document.getElementsByTagName('body')[0].style.webkitFontSizeAdjust=
'%f%%'",Slider.value];
    //改变字体的大小
    [webView stringByEvaluatingJavaScriptFromString:str1];
}
```

【代码解析】

本实例关键功能是网页视图中网页元素的交互。下面就是这个知识点的详细讲解。

想要实现网页视图中网页元素的交互,需要使用 UIWebView 的 stringByEvaluatingJavaScriptFromString:方法,其语法形式如下:

```
- (NSString *)stringByEvaluatingJavaScriptFromString:(NSString *)script;
```

其中, (NSString *)script 表示运行的脚本。在本实例中就使用了 stringByEvaluatingJavaScriptFromString:方法实现了改变字体的大小,程序代码如下:

```
[webView stringByEvaluatingJavaScriptFromString:str1];
```

其中, str1 表示运行的脚本。

实例 64 网页视图的背景透明化

【实例描述】

本实例的功能是让网页视图的背景实现透明化的效果。运行效果如图 5.7 所示。



图 5.7 运行效果

【实现过程】

- (1) 创建一个项目,命名为“网页视图的背景透明化”。
- (2) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件,编写代码,实现插座变量的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    IBOutlet UIWebView *webView;
}
@end
```

- (4) 打开 Main.storyboard 文件,对 View Controller 视图控制器的设计界面进行设计,

效果如图 5.8 所示。

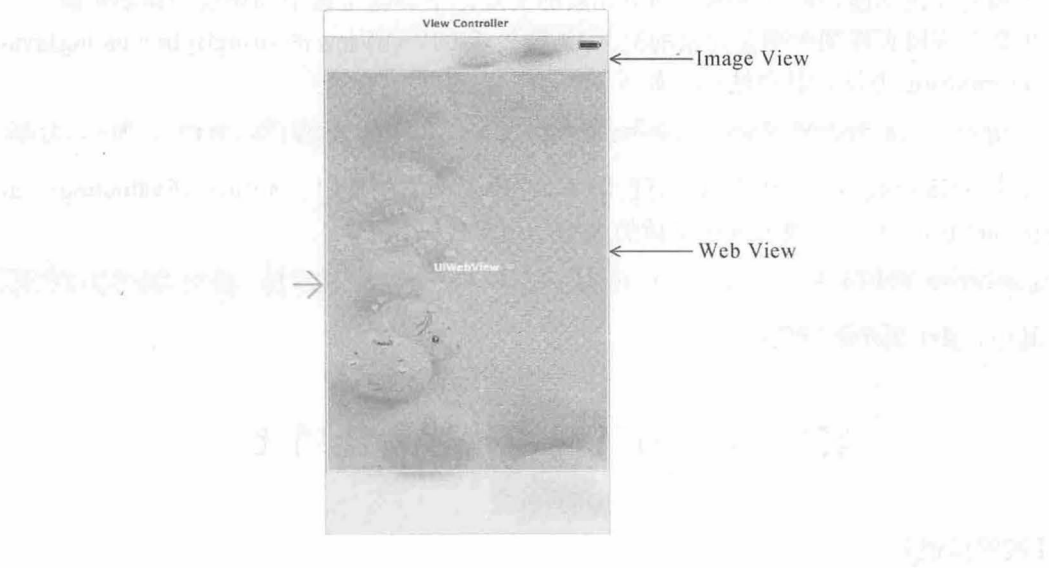


图 5.8 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-6 所示。

表 5-6 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 1.jpg	
Web View		与插座变量 webView 关联

(5) 打开 ViewController.m 文件，编写代码，实现加载网址以及背景透明化的效果。
程序代码如下：

```
- (void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //设置背景透明
    webView.backgroundColor=[UIColor clearColor];
    webView.opaque=NO;
    NSURL *url=[NSURL URLWithString:@"http://www.baidu.com"];
    //创建 NSURLRequest 对象，传递网址
    NSURLRequest *request=[NSURLRequest requestWithURL:url];
    //加载网址对应的网页内容
    [webView loadRequest:request];
}
```

【代码解析】

本实例关键功能是背景透明。下面就是这个知识点的详细讲解。

在本实例中网页视图的背景透明是通过使用 UIView 的 opaque 属性，它表示当前的 UIView 是否不透明，其语法形式如下：

```
@property(n nonatomic, getter=isOpaque) BOOL opaque;
```

其中, 该属性为 BOOL 值。在本实例中的代码如下:

```
webView.opaque=NO;
```

实例 65 网页的下拉刷新

【实例描述】

在 QQ 空间、微博等网页中, 在向下拉时, 会出现刷新的效果, 本实例就为读者实现网页的下拉刷新效果。运行效果如图 5.9 所示。



图 5.9 运行效果

【实现过程】

- (1) 创建一个项目, 命名为“网页的下拉刷新”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 RefreshView 类。
- (4) 打开 RefreshView.h 文件, 编写代码, 实现头文件、宏定义、数据结构的定义、协议、实例变量、对象、属性以及方法的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@class RefreshView;
//宏定义
#define TEXT_COLOR [UIColor colorWithRed:87.0/255.0 green:108.0/255.0
blue:137.0/255.0 alpha:1.0]
#define FLIP_ANIMATION_DURATION 0.18f
//数据结构的定义
typedef enum{
    EGOOPullRefreshPulling = 0,
    EGOOPullRefreshNormal,
    EGOOPullRefreshLoading,
} PullRefreshState;
//协议
@protocol RefreshTableHeaderDelegate
- (void)RefreshTableHeaderDidTriggerRefresh:(RefreshView*)view;
//调用加载网页的方法
```



```

- (BOOL)RefreshTableHeaderDataSourceIsLoading: (RefreshView*)view;
//获取是否重新加载

@optional
- (NSDate*)RefreshTableHeaderDataSourceLastUpdated: (RefreshView*)view;
//获取日期

@end

@interface RefreshView : UIView{
    __unsafe_unretained id _delegate;
    PullRefreshState _state;
    UILabel *_lastUpdatedLabel; //声明标签对象
    UILabel *_statusLabel;
    CALayer *_arrowImage;
    UIActivityIndicatorView *_activityView; //声明加载指示器对象
}

@property(n nonatomic, assign) id <RefreshTableHeaderDelegate> delegate;
//方法
- (void)refreshLastUpdatedDate; //刷新最后更新的时间
- (void)RefreshScrollViewDidScroll: (UIScrollView *)scrollView;
- (void)RefreshScrollViewDidEndDragging: (UIScrollView *)scrollView;
- (void)RefreshScrollViewDidEndDraggingWithRefreshButton: (UIScrollView *)scrollView;
- (void)RefreshScrollViewDataSourceDidFinishedLoading: (UIScrollView *)scrollView;
- (void)setState: (PullRefreshState) aState; //设置状态
@end

```

(5) 打开 RefreshView.m 文件，编写代码，实现刷新视图的绘制以及刷新功能。使用的方法如表 5-7 所示。

表 5-7 RefreshView.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
refreshLastUpdatedDate	刷新最后更新的时间
setState:	设置状态
RefreshScrollViewDidScroll:	网页视图滚动时调用，判断状态
RefreshScrollViewDidEndDragging:	在网页视图滚动结束时调用，实现显示刷新视图的加载过程
RefreshScrollViewDataSourceDidFinishedLoading:	隐藏刷新视图

绘制刷新视图需要使用 initWithFrame:、setState:方法。其中，initWithFrame:方法实现刷新视图的初始化功能。程序代码如下：

```

- (id)initWithFrame:(CGRect)frame {
    if (self = [super initWithFrame:frame]) {
        self.autoresizingMask = UIViewAutoresizingFlexibleWidth;
        self.backgroundColor = [UIColor colorWithRed:226.0/255.0
            green:231.0/255.0 blue:237.0/255.0 alpha:1.0]; //设置背景颜色
        //创建并设置标签对象
        UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(0.0f,
            frame.size.height - 30.0f, self.frame.size.width, 20.0f)];
        label.autoresizingMask = UIViewAutoresizingFlexibleWidth;
        label.font = [UIFont systemFontOfSize:12.0f]; //设置字体
        label.textColor = TEXT_COLOR;
        label.shadowColor = [UIColor colorWithWhite:0.9f alpha:1.0f];
        //设置阴影颜色
    }
}

```

```

label.shadowOffset = CGSizeMake(0.0f, 1.0f);
label.backgroundColor = [UIColor clearColor]; //设置背景颜色
label.textAlignment = NSTextAlignmentCenter;
[self addSubview:label]; //添加标签对象
_lastUpdatedLabel=label;
label = [[UILabel alloc] initWithFrame:CGRectMake(0.0f, frame.size.
height - 48.0f, self.frame.size.width, 20.0f)];
label.autoresizingMask = UIViewAutoresizingFlexibleWidth;
label.font = [UIFont boldSystemFontOfSize:13.0f];
label.textColor = TEXT_COLOR; //设置文本颜色
label.shadowColor = [UIColor colorWithWhite:0.9f alpha:1.0f];
label.shadowOffset = CGSizeMake(0.0f, 1.0f);
label.backgroundColor = [UIColor clearColor];
label.textAlignment = NSTextAlignmentCenter; //设置对齐方式
[self addSubview:label];
_statusLabel=label;
//创建并设置图层对象
CALayer *layer = [CALayer layer];
layer.frame = CGRectMake(25.0f, frame.size.height - 65.0f, 30.0f,
55.0f); //设置框架
layer.contentsGravity = kCAGravityResizeAspect;
layer.contents = (id)[UIImage imageNamed:@"1.png"].CGImage; //显示图像

#if __IPHONE_OS_VERSION_MAX_ALLOWED >= 40000
if ([[UIScreen mainScreen] respondsToSelector:@selector(scale)]) {
    layer.contentsScale = [[UIScreen mainScreen] scale]; //设置内容的缩放值
}
#endif
[[self layer] addSublayer:layer]; //添加图层对象
_arrowImage=layer;
//创建并设置加载视图对象
UIActivityIndicatorView *view = [[UIActivityIndicatorView alloc]
initWithActivityIndicatorStyle:UIActivityIndicatorViewStyleGray];
view.frame = CGRectMake(25.0f, frame.size.height - 38.0f, 20.0f,
20.0f); //设置框架
[self addSubview:view];
_activityView = view;
[self setState:EGOOPullRefreshNormal]; //设置状态
}
return self;
}

```

setState:方法实现对当前的状态进行设置。程序代码如下:

```

- (void)setState:(PullRefreshState)aState{
    switch (aState) {
        case EGOOPullRefreshPulling:
            _statusLabel.text = NSLocalizedString(@"Release to refresh...",
@"Release to refresh status"); //设置文本内容
            [CATransaction begin];
            [CATransaction
setAnimationDuration:FLIP_ANIMATION_DURATION];
            _arrowImage.transform = CATransform3DMakeRotation((M_PI /
180.0) * 180.0f, 0.0f, 0.0f, 1.0f); //旋转
            [CATransaction commit];
            break;
        case EGOOPullRefreshNormal:
            //判断_state 是否为 EGOOPullRefreshPulling

```

```

        if (_state == EGOOPullRefreshPulling) {
            [CATransaction begin];
            [CATransaction setAnimationDuration:FLIP_ANIMATION_
            DURATION]; //设置动画时间
            _arrowImage.transform = CATransform3DIdentity; //设置改变
            [CATransaction commit];
        }
        _statusLabel.text = NSLocalizedString(@"Pull down to refresh
        ...", @"Pull down to refresh status"); //设置文本内容
        [_activityView stopAnimating];
        [CATransaction begin];
        [CATransaction setValue:(id)kCFBooleanTrue forKey:
        kCATransactionDisableActions];
        _arrowImage.hidden = NO; //隐藏图像
        _arrowImage.transform = CATransform3DIdentity;
        [CATransaction commit];
        [self refreshLastUpdatedDate];
        break;
    case EGOOPullRefreshLoading:
        _statusLabel.text = NSLocalizedString(@"Loading...", @"Loading
        Status"); //设置文本内容
        [_activityView startAnimating];
        [CATransaction begin];
        [CATransaction setValue:(id)kCFBooleanTrue forKey:
        kCATransactionDisableActions];
        _arrowImage.hidden = YES; //隐藏图像
        [CATransaction commit];
        break;
    default:
        break;
    }
    _state = aState;
}

```

刷新视图的刷新需要使用 refreshLastUpdatedDate、RefreshScrollViewDidScroll:、RefreshScrollViewDidEndDragging:和 RefreshScrollViewDataSourceDidFinishedLoading:方法实现。其中，refreshLastUpdatedDate 方法实现对网页视图最后更新的时间进行刷新。程序代码如下：

```

- (void)refreshLastUpdatedDate {
    if ([_delegate respondsToSelector:@selector(RefreshTableHeaderData
    SourceLastUpdated:)]) {
        NSDate *date = [_delegate RefreshTableHeaderDataSourceLastUpdated:
        self]; //创建日期对象
        NSDateFormatter *formatter = [[NSDateFormatter alloc] init];
        [formatter setAMSymbol:@"AM"];
        [formatter setPMSymbol:@"PM"];
        [formatter setDateFormat:@"MM/dd/yyyy hh:mm:a"]; //设置日期格式
        _lastUpdatedLabel.text = [NSString stringWithFormat:@"Last Updated:
        %@", [formatter stringFromDate:date]]; //设置文本内容
        [[NSUserDefaults standardUserDefaults] setObject:_lastUpdatedLabel.
        text forKey:@"EGORefreshTableView_LastRefresh"];
        [[NSUserDefaults standardUserDefaults] synchronize];
    } else {
        _lastUpdatedLabel.text = nil; //设置文本内容
    }
}

```

RefreshScrollViewDidScroll:方法在网页视图滚动时调用,实现对当前状态的判断以及设置。程序代码如下:

```
- (void)RefreshScrollViewDidScroll:(UIScrollView *)scrollView {
    if (_state == EGOOPullRefreshLoading) {
        CGFloat offset = MAX(scrollView.contentOffset.y * -1, 0);
        offset = MIN(offset, 60);
        scrollView.contentInset = UIEdgeInsetsMake(offset, 0.0f, 0.0f, 0.0f);
        //设置可滚动区域显示的坐标
    } else if (scrollView.isDragging) {
        BOOL _loading = NO;
        //判断是否实现了 RefreshTableHeaderDataSourceIsLoading:方法
        if ([_delegate respondsToSelector:@selector(RefreshTableHeaderDataSourceIsLoading:)]) {
            _loading = [_delegate RefreshTableHeaderDataSourceIsLoading:self];
        }
        if (_state == EGOOPullRefreshPulling && scrollView.contentOffset.y > -65.0f && scrollView.contentOffset.y < 0.0f && !_loading) {
            [self setState:EGOOPullRefreshNormal]; //设置状态
        } else if (_state == EGOOPullRefreshNormal && scrollView.contentOffset.y < -65.0f && !_loading) {
            [self setState:EGOOPullRefreshPulling]; //设置状态
        }
        if (scrollView.contentInset.top != 0) {
            scrollView.contentInset = UIEdgeInsetsZero;
            //设置可滚动区域显示的坐标
        }
    }
}
```

RefreshScrollViewDidEndDragging:方法实现显示刷新视图。程序代码如下:

```
- (void)RefreshScrollViewDidEndDragging:(UIScrollView *)scrollView {
    BOOL _loading = NO;
    //判断是否实现了 RefreshTableHeaderDataSourceIsLoading:方法
    if ([_delegate respondsToSelector:@selector(RefreshTableHeaderDataSourceIsLoading:)]) {
        _loading = [_delegate RefreshTableHeaderDataSourceIsLoading:self];
    }
    if (scrollView.contentOffset.y <= - 65.0f && !_loading) {
        //判断是否实现了 RefreshTableHeaderDidTriggerRefresh:方法
        if ([_delegate respondsToSelector:@selector(RefreshTableHeaderDidTriggerRefresh:)]) {
            [_delegate RefreshTableHeaderDidTriggerRefresh:self];
        }
        [self setState:EGOOPullRefreshLoading]; //设置状态
        [UIView beginAnimations:nil context:NULL];
        [UIView setAnimationDuration:0.2]; //设置动画的持续时间
        scrollView.contentInset = UIEdgeInsetsMake(60.0f, 0.0f, 0.0f, 0.0f);
        //设置可滚动区域显示的坐标
        [UIView commitAnimations];
    }
}
```

RefreshScrollViewDataSourceDidFinishedLoading:实现隐藏刷新视图。程序代码如下:

```
- (void)RefreshScrollViewDataSourceDidFinishedLoading:(UIScrollView *)
```

```
scrollView {
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:.3];           //设置动画的持续时间
    [scrollView setContentInset:UIEdgeInsetsMake(0.0f, 0.0f, 0.0f, 0.0f)];
    [UIView commitAnimations];
    [self setState:EGOOPullRefreshNormal];
}
```

(6) 打开 ViewController.h 文件，编写代码实现，实现头文件、遵守协议、对象、实例变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "RefreshView.h"           //头文件
@interface ViewController : UIViewController<UIWebViewDelegate,
UIScrollViewDelegate,
RefreshTableHeaderDelegate >{
    //对象
    RefreshView * _refreshView;
    IBOutlet UIWebView *web;
    BOOL _reloading;              //布尔类型的实例变量
}
@end
```

(7) 打开 Main.storyboard 文件，从视图库中拖动 Web View 网页视图到设计界面中，将声明的插座变量与此视图进行关联。

(8) 打开 ViewController.m 文件，编写代码，实现网页的下拉刷新效果。使用的方法如表 5-8 所示。

表 5-8 ViewController.m 文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
loadPage	加载网页
webViewDidStartLoad:	网页视图开始加载
webViewDidFinishLoad:	网页视图结束加载
webView:didFailLoadWithError:	网页视图加载失败
scrollViewDidScroll:	滚动视图
scrollViewDidEndDragging:willDecelerate:	用户拖曳并且停止
scrollViewDidEndScrollingAnimation:	视图滚动并伴有动画
RefreshTableHeaderDidTriggerRefresh:	调用加载网页的方法
RefreshTableHeaderDataSourceIsLoading:	获取是否重新加载
RefreshTableHeaderDataSourceLastUpdated:	获取日期

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewDidLoad 方法实现初始化的设置。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    web.delegate = self;
    web.scrollView.delegate = self;           //设置委托
    [self loadPage];
    //初始化 refreshView，添加到 web 的 scrollView 子视图中
```



```

if (_refreshView == nil) {
    _refreshView = [[RefreshView alloc] initWithFrame:CGRectMake(0,
        0-web.scrollView.bounds.size.height, web.scrollView.frame.size.
        width, web.scrollView.bounds.size.height)];           //创建刷新视图
    _refreshView.delegate = self;
    [web.scrollView addSubview:_refreshView];
}
[_refreshView refreshLastUpdatedDate];
}

```

loadPage 方法实现网页视图的加载。程序代码如下：

```

- (void)loadPage {
    NSURL *url = [[NSURL alloc] initWithString:@"http://www.baidu.com"];
    NSURLRequest *request = [[NSURLRequest alloc] initWithURL:url];
    [web loadRequest:request];                               //加载
}

```

【代码解析】

由于本实例中的代码和方法非常多，为了方便读者的阅读，笔者绘制了一些执行流程图，如图 5.11 和图 5.12 所示。其中，进度条的显示，使用了 initWithFrame:、refreshLastUpdatedDate、setState:、RefreshScrollViewDidScroll:、RefreshScrollViewDataSourceDidFinishedLoading:、viewDidLoad、loadPage、webViewDidStartLoad:、webViewDidFinishLoad:、scrollViewDidScroll:、RefreshTableHeaderDataSourceLastUpdated:方法共同实现，它们的执行流程如图 5.10 所示。

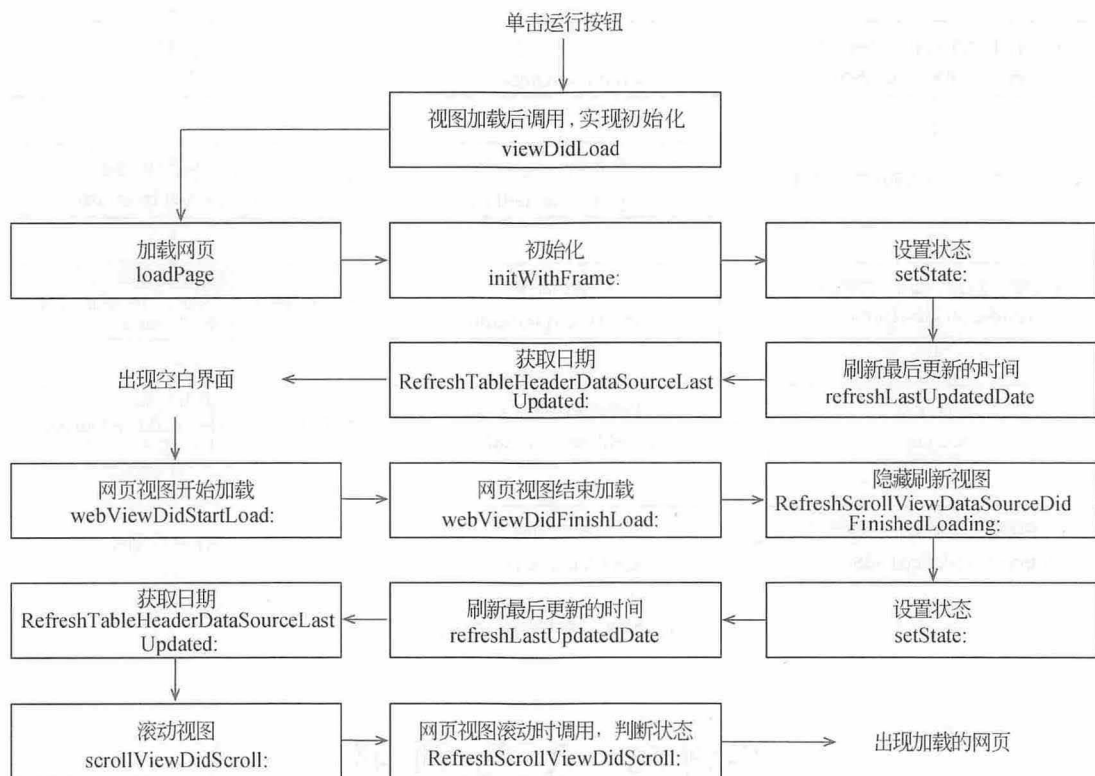


图 5.10 程序执行流程

当进行下拉时，会出现刷新视图，并且进行刷新，此时需要使用 refreshLastUpdatedDate、RefreshScrollViewDidScroll:、setState:、RefreshScrollViewDidEndDragging:、RefreshScroll-

ViewDataSourceDidFinishedLoading:、loadPage、webViewDidStartLoad:、webViewDidFinishLoad:、scrollViewDidScroll:、scrollViewDidEndDragging:willDecelerate:、RefreshTableHeaderDidTriggerRefresh:、RefreshTableHeaderDataSourceIsLoading:方法共同实现。它们的执行流程如图 5.11 所示。

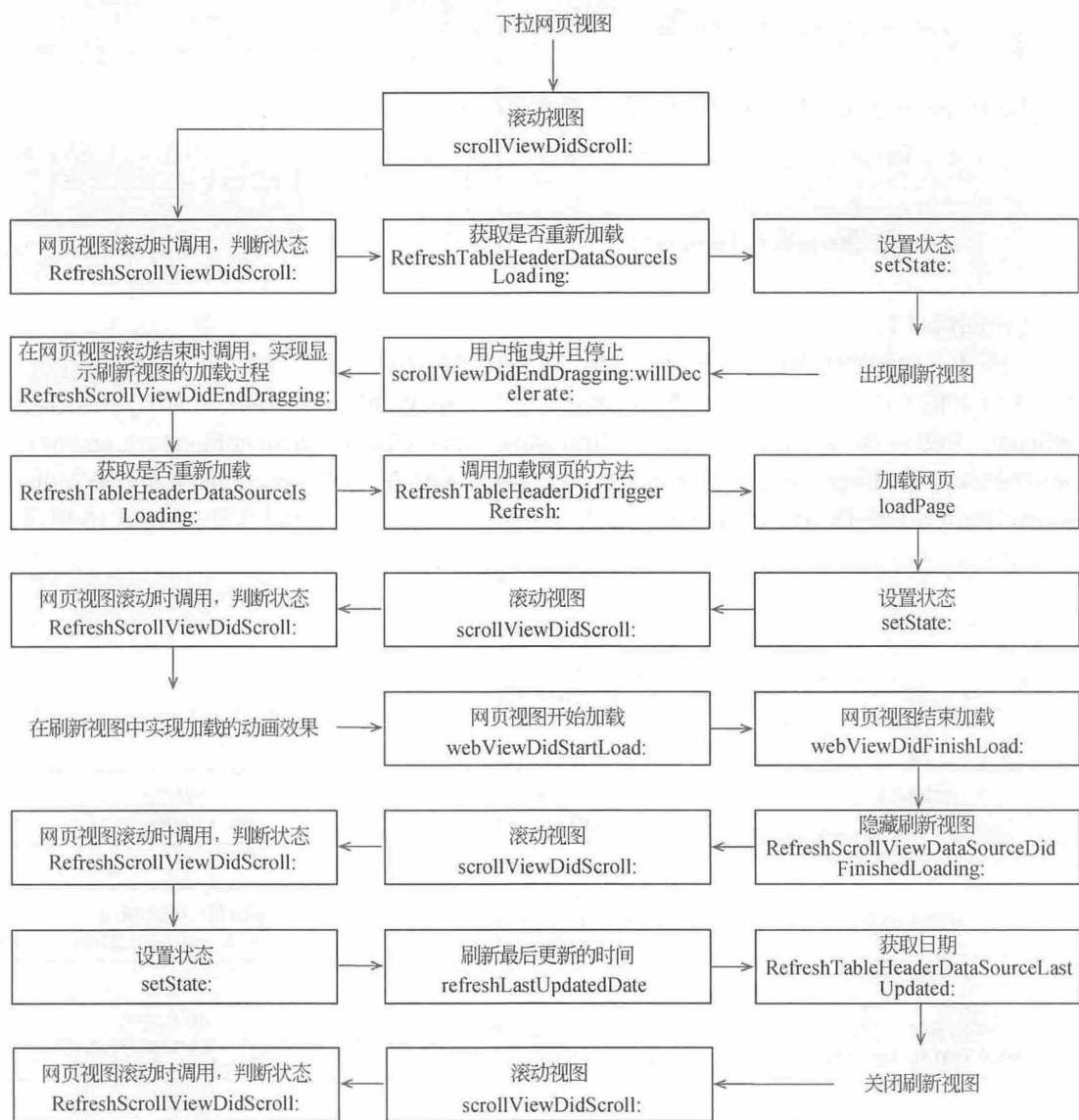


图 5.11 程序执行流程

实例 66 天气预报

【实例描述】

本实例实现的功能是显示北京的天气预报信息。其中包含天气状况、最高温度、最低

温度、日期等。运行效果如图 5.12 所示。

【实现过程】

- (1) 创建一个项目，命名为“天气预报”。
- (2) 添加图像 1.jpg、2.png~15.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现插座变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
IBOutlet UIImageView *imageView;
//声明关于标签的插座变量
IBOutlet UILabel *label1;
IBOutlet UILabel *label2;
IBOutlet UILabel *label3;
IBOutlet UILabel *label4;
}
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 5.13 所示。

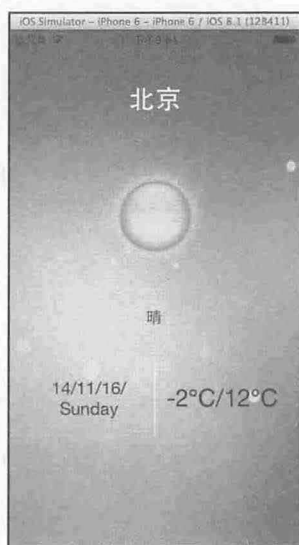


图 5.12 运行效果

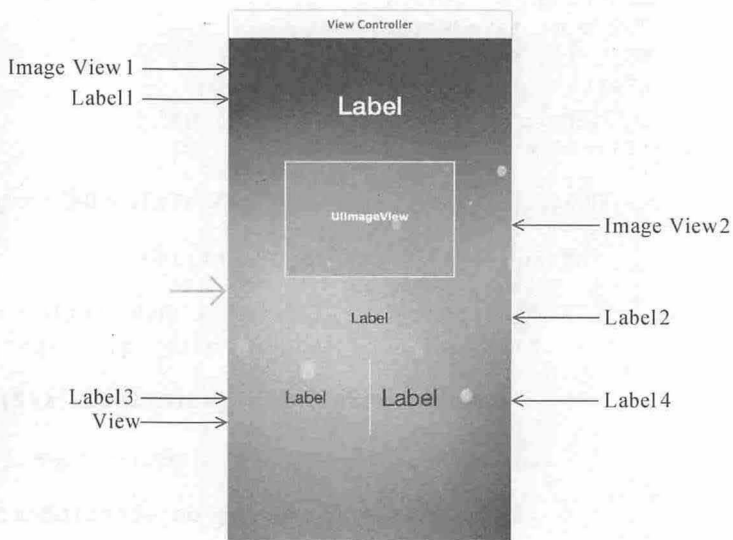


图 5.13 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-9 所示。

表 5-9 视图、控件设置

视图、控件	属性设置	其他
Image View1	Image: 1.jpg	
Image View2		与插座变量 imageView 关联
Label1	Color: 白色 Font: System Bold 29.0 Alignment: 居中	与插座变量 label1 关联
Label2	Alignment: 居中	与插座变量 label4 关联

续表

视图、控件	属 性 设 置	其 他
Label3	Font: System 19.0 Alignment: 居中 Lines: 2	与插座变量 label3 关联
Label4	Font: System 27.0	与插座变量 label4 关联
View		

(5) 打开 ViewController.m 文件，编写代码，实现显示天气预报的功能。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    NSURL *URL = [NSURL URLWithString:@"http://www.weather.com.cn/data/cityinfo/101010100.html"];
    NSError *error;
    NSString *stringFromFileAtURL = [[NSString alloc] initWithContentsOfURL:
    URL encoding:NSUTF8StringEncoding error:&error]; //创建字符串对象
    NSString *strTempL;
    NSString *strTempH;
    NSString *strWeather;
    NSString *name;
    NSString *fileName=@"8.png";
    //判断 stringFromFileAtURL 是否不为空
    if(stringFromFileAtURL !=nil)
    {
        NSArray *strarray = [stringFromFileAtURL componentsSeparatedByString:
        @"\n"];
        for(int i=0;i<strarray.count;i++)
        {
            NSString *str = [strarray objectAtIndex:i];
            if(YES == [str isEqualToString:@"city"]) //城市
            {
                name = [strarray objectAtIndex:i+2];
            }
            if(YES == [str isEqualToString:@"temp1"]) //最高温度
            {
                strTempH = [strarray objectAtIndex:i+2];
            }
            else if(YES == [str isEqualToString:@"temp2"]) //最低温度
            {
                strTempL = [strarray objectAtIndex:i+2];
            }
            else if(YES == [str isEqualToString:@"weather"])//天气
            {
                strWeather = [strarray objectAtIndex:i+2];
            }
        }
        //在 strWeather 这个字符串中搜索对应的内容，判断其是否存在
        if(NSNotFound != [strWeather rangeOfString:@"晴"].location)
        {
            fileName =[[NSString alloc] initWithFormat:@"%s",@"8.png"];
            //创建字符串对象
        }
        if(NSNotFound != [strWeather rangeOfString:@"多云"].location)
```

```

{
    fileName = [[NSString alloc] initWithFormat:@"%@", @"5.png"];
    //创建字符串对象
}
.....
if(NSNotFound != [strWeather rangeOfString:@"阴转晴"].location)
{
    fileName = [[NSString alloc] initWithFormat:@"%@", @"12.png"];
    //创建字符串对象
}
label1.text= [[NSString alloc] initWithFormat:@"%@", name];
NSString *sweather = [[NSString alloc] initWithFormat:@"%d/%d",
strTempL, strTempH];
label2.text=sweather;
label4.text=[[NSString alloc] initWithFormat:@"%d", strWeather];
//设置标签的文本内容
}
imageView.image = [UIImage imageNamed:fileName];
//日期格式的转换
NSDateFormatter *f=[[NSDateFormatter alloc] init];
[f setDateFormat:@"%y/MM/dd"]; //设置日期的格式
NSDate *date=[NSDate date];
NSString *str=[f stringFromDate:date];
[f setDateFormat:@"%EEEE"]; //设置日期的格式
NSString *str1=[f stringFromDate:date];
label3.text=[[NSString alloc] initWithFormat:@"%d/%d", str, str1];
}

```

【代码解析】

本实例关键功能是相关信息的获取以及图像的显示。下面依次讲解这两个知识点。

1. 相关信息的获取

在本实例中，为了获取城市名称、最高温度、最低温度这些信息，首先，需要创建一个 NSURL 对象，它其实就是一个网站的网址。其次将创建的 NSURL 对象放入到 NSString 字符串对象中，代码如下：

```

NSString *stringFromFileAtURL = [[NSString alloc] initWithContentsOfURL:URL
encoding:NSUTF8StringEncoding error:&error];

```

然后将 NSString 对象转换为数组对象，代码如下：

```

NSArray *strarray = [stringFromFileAtURL componentsSeparatedByString:
@"\""];

```

最后遍历数组对象，获取数组中的元素，进行判断，以获取程序名称为例，就将数组中获取的元素与“city”进行判断，如果相等，就获取此对象位置加2的元素。代码如下：

```

for(int i=0;i<strarray.count;i++)
{
    NSString *str = [strarray objectAtIndex:i];
    if(YES == [str isEqualToString:@"city"])//城市
    {
        name = [strarray objectAtIndex:i+2];
    }
    .....
}

```


2. 图像的显示

在本实例中，图像视图中会显示指定网址中关于天气的相关图片，需要在 `strWeather`（用来放置天气的字符串对象）字符串中搜索指定字符串。如果此字符串存在，就创建一个新的字符串。代码如下：

```
if(NSNotFound != [strWeather rangeOfString:@"晴"].location)
{
    fileName = [NSString alloc] initWithFormat:@"%s", @"8.png"];
}
.....
```

最后指定图像视图的图像，图像的内容就是指定的相应的字符串。代码如下：

```
imageView.image = [UIImage imageNamed:fileName];
```

实例 67 城市地理信息查询

【实例描述】

iOS 5 之后的版本自带了 JSON 数据解析的类。本实例就使用此类实现解析关于城市地理信息的 JSON 数据。在文本框中输入数据城市后就会显示相关的信息。运行效果如图 5.14 所示。

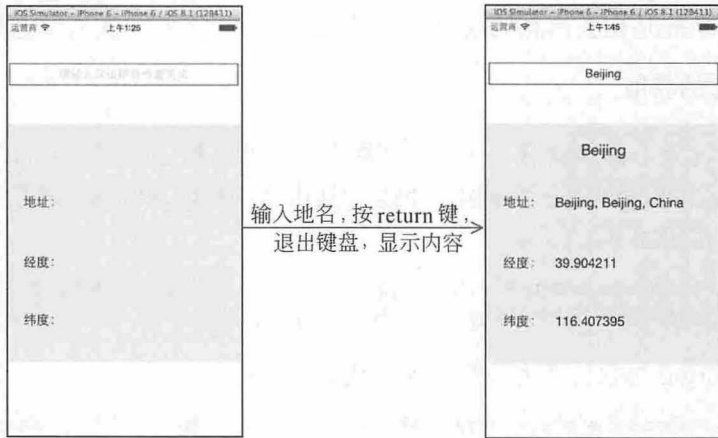


图 5.14 运行效果

【实现过程】

(1) 创建一个项目，命名为“城市地理信息查询”。

(2) 打开 `ViewController.h` 文件，编写代码，实现插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    IBOutlet UITextField *tf;
    //声明关于标签的插座变量
```

```

IBOutlet UILabel *label1;
IBOutlet UILabel *label2;
IBOutlet UILabel *label3;
IBOutlet UILabel *label4;
}
- (IBAction)Search:(id) sender;
@end

```

(3) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 5.15 所示。

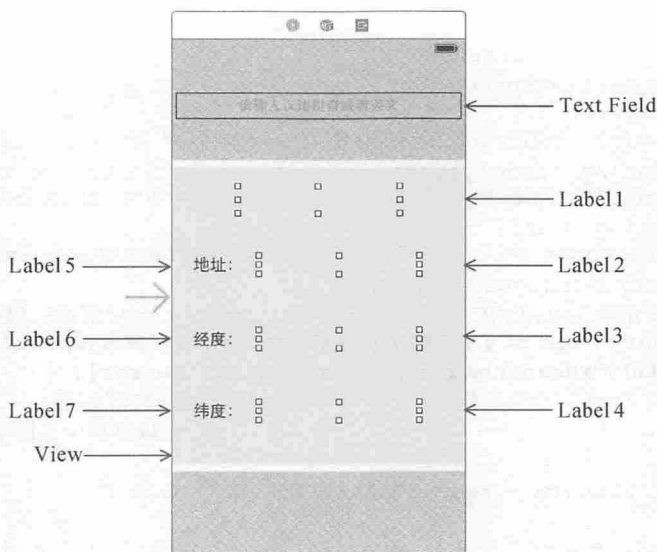


图 5.15 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-10 所示。

表 5-10 视图、控件设置

视图、控件	属性设置	其它
设计界面	Background: 浅灰色	
Text Field	Alignment: 居中 Placeholder: 请输入汉语拼音或者英文 Border Style: 线框风格	与插座变量 tf 关联 与动作 Search:关联 Did End On Exit 与动作 Search:关联
Label1	Text: (空) Font: System 21.0	与插座变量 label1 关联
Label2	Text: (空) Font: System 18.0	与插座变量 label2 关联
Label3	Text: (空) Font: System 18.0	与插座变量 label3 关联
Label4	Text: Font: System 18.0	与插座变量 label4 关联
Label5	Text: 地址: Font: System 18.0	

续表

视图、控件	属 性 设 置	其 它
Label6	Text: 经度: Font: System 18.0	
Label7	Text: 纬度: Font: System 18.0	
View	Background: 浅紫色	

(4) 打开 ViewController.m 文件, 编写代码, 实现 JSON 数据的解析并显示功能。程序代码如下:

```

- (IBAction)Search:(id)sender {
    [tf resignFirstResponder]; //关闭键盘
    NSError *error;
    NSString *url=[NSString stringWithFormat:@"http://maps.googleapis.
com/maps/api/geocode/json?address=%@&sensor=true",tf.text];
    NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL
URLWithString:url]];
    NSData *response = [NSURLConnection sendSynchronousRequest:request
returningResponse:nil error:nil];
    //iOS5 自带解析类 NSJSONSerialization 从 response 中解析出数据放到字典中
    NSDictionary *dic = [NSJSONSerialization JSONObjectWithData:response
options:NSJSONReadingMutableLeaves error:&error];
    NSArray *results = [dic objectForKey:@"results"];
    //获取字典中关键字为 results 的值

    //遍历数组
    for (NSDictionary * resultDetailDic in results)
    {
        NSArray *name=[resultDetailDic objectForKey:@"address_
components"];
        NSMutableArray *array=[NSMutableArray array];
        //变量数组
        for (NSDictionary * resul in name){
            NSString *nam=[resul objectForKey:@"long_name"];
            [array addObject:nam]; //添加对象
        }
        NSString *address=[resultDetailDic objectForKey:@"formatted_
address"];
        NSDictionary * locationDic=[[resultDetailDic objectForKey:@"
geometry"] objectForKey:@"location"];
        NSString * lat=[locationDic objectForKey:@"lat"];
        //获取字典中关键字为 lat 的值
        NSString * lng=[locationDic objectForKey:@"lng"];
        dispatch_async(dispatch_get_main_queue(), ^{
            //设置标签的文本内容
            label1.text=[NSString stringWithFormat:@"%@",[array firstObject]];
            label2.text=[NSString stringWithFormat:@"%@",address];
            label3.text=[NSString stringWithFormat:@"%@",lat];
            label4.text=[NSString stringWithFormat:@"%@",lng];
        });
    }
}

```

【代码解析】

本实例关键功能是解析 Json。下面就是这个知识点的详细讲解。早在 iOS 5 中, 就添

加了 NSJSONSerialization 类来对 Json 进行解析,本实例中就是使用了 NSJSONSerialization 类对 Json 进行的解析,步骤如下:

1. 放入缓存中

要解析数据,首先要将解析的 Json 数据放入到缓存中,在本实例中,实现此步骤的代码如下:

```
NSData *response = [NSURLConnection sendSynchronousRequest:request
returningResponse:nil error:nil];
```

2. 解析

解析数据,需要使用 NSJSONSerialization 的 JSONObjectWithData:options:error:方法实现,它的功能是使用缓冲区数据来解析 Json 属性,其语法形式如下:

```
+ (id)JSONObjectWithData:(NSData *)data options:(NSJSONReadingOptions)opt
error:(NSError **)error;
```

其中,(NSData *)data 表示一个包含 Json 数据的数据对象;(NSJSONReadingOptions)opt 表示读取数据并创建 Foundation 对象的选项,此选项包含 3 个,如表 5-11 所示。(NSError **)error 表示如果有错误发生就返回一个 NSError 对象。

表 5-11 opt选项

方 法	功 能
NSJSONReadingMutableContainers	指定可变的数据和字典对象
NSJSONReadingMutableLeaves	指定 NSString 是可变类型的
NSJSONReadingAllowFragments	指定解析器应该允许不属于的 NSArray 或 NSDictionary 中的实例顶层对象

3. 遍历

最后使用遍历的方法获取关键字对应的值,代码如下:

```
for (NSDictionary * resultDetailDic in results)
{
    NSArray *name=[resultDetailDic objectForKey:@"address_components"];
    NSMutableArray *array=[NSMutableArray array];
    for (NSDictionary * resul in name){
        NSString *nam=[resul objectForKey:@"long_name"];
        [array addObject:nam];
    }
    NSString *address=[resultDetailDic objectForKey:@"formatted_
address"];
    NSDictionary * locationDic=[[resultDetailDic objectForKey:@"geometry"]
objectForKey:@"location"];
    NSString * lat=[locationDic objectForKey:@"lat"];
    NSString * lng=[locationDic objectForKey:@"lng"];
    dispatch_async(dispatch_get_main_queue(), ^{
        label1.text=[NSString stringWithFormat:@"%@",[array firstObject]];
        label2.text=[NSString stringWithFormat:@"%@",address];
        label3.text=[NSString stringWithFormat:@"%@",lat];
        label4.text=[NSString stringWithFormat:@"%@",lng];
    });
}
```


实例 68 滑动网页时，隐藏工具栏

【实例描述】

本实例实现的功能是滑动网页时，隐藏工具栏的运行。当用户滑动网页时，工具栏就会隐藏；当停止滑动时，隐藏的工具栏就会显示。运行效果如图 5.16 所示。

【实现过程】

- (1) 创建一个项目，命名为“滑动网页时，隐藏工具栏”。
- (2) 打开 ViewController.h 文件，编写代码，实现遵守协议、插座变量的声明。程序代码如下所示：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController<UIWebViewDelegate,
    UIGestureRecognizerDelegate>{
    IBOutlet UIWebView *webview;                //声明关于网页视图的插座变量
    IBOutlet UIToolbar *myToolbar;              //声明关于工具栏的插座变量
}
@end
```

- (3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 5.17 所示。



图 5.16 运行效果

图 5.17 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-12 所示。

表 5-12 视图、控件设置

视图、控件	属性设置	其他
Web View		与插座变量 webview 关联
Toolbar	Style: Black Bar Tint: 黑色	与插座变量 myToolbar 关联

(4) 打开 ViewController.m 文件, 编写代码, 实现在滑动网页时将工具栏隐藏的功能。使用的方法如表 5-13 所示。

表 5-13 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
handleGesture:	拖动网页视图
gestureRecognizer:shouldRecognizeSimultaneously WithGestureRecognizer:	同时接收两个事件
gestureRecognizer:shouldReceiveTouch:	询问委托是否允许手势接收者接收一个触摸对象
gestureRecognizerShouldBegin:	询问一个手势接收者是否应该开始解释执行一个触摸接收事件

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, viewDidLoad 方法实现网页视图的设置以及工具栏的创建和设置。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [webView loadRequest:[NSURLRequest requestWithURL:[NSURL alloc]
initWithString:@"http://www.hao123.com"]]; //加载内容
    webView.scalesPageToFit = TRUE;
    [webView setUserInteractionEnabled: YES]; //是否支持交互
    [webView setDelegate: self]; //委托
    [webView setOpaque: YES]; //透明
    //创建并设置工具栏中的按钮
    UIBarButtonItem *button1 = [[UIBarButtonItem alloc] initWithTitle:
@"首页" style:UIBarButtonItemStylePlain target:nil action:nil];
    UIBarButtonItem *button2 = [[UIBarButtonItem alloc] initWithTitle:
@"通讯录" style:UIBarButtonItemStyleBordered target:nil action:nil];
    [button2 setWidth:100.0]; //设置宽度
    //创建按钮条目
    UIBarButtonItem *button3 = [[UIBarButtonItem alloc] initWithImage:
[UIImage imageNamed:@"apple_icon.png"] style:UIBarButtonItemStyle
Bordered target:nil action:nil];
    UIBarButtonItem *flexButton = [[UIBarButtonItem alloc] initWithBar
ButtonSystemItem:UIBarButtonSystemItemFlexibleSpace target:nil action:
nil];
    UIBarButtonItem *trashButton = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:
UIBarButtonItemSystemItemTrash target:nil action:nil];
    NSArray *buttons = [[NSArray alloc] initWithObjects:button1,button2,
button3, flexButton,trashButton,nil];
    [myToolbar setItems:buttons animated:YES]; //设置条目
    [myToolbar setTag:999]; //设置 tag 值
    //创建并设置移动的手势识别器对象
    UIPanGestureRecognizer* singlePan = [[UIPanGestureRecognizer alloc]
initWithTarget:self action:@selector(handleGesture)];
    [self.view addGestureRecognizer:singlePan]; //添加手势识别器对象
    singlePan.delegate = self;
    singlePan.cancelsTouchesInView = NO;
}
```

handleGesture:方法实现滚动网页视图时, 工具栏的显示和隐藏功能。程序代码如下:

```

- (void)handleGesture:(UIGestureRecognizer *)gestureRecognizer
{
    switch (gestureRecognizer.state) {
        case UIGestureRecognizerStateEnded:{
            CGContextRef context = UIGraphicsGetCurrentContext();
                                                    //创建图形上下文
            [UIView beginAnimations:nil context:context];
            [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
            [UIView setAnimationDuration:1.0];           //设置动画持续时间
            [[self.view viewWithTag:999] setAlpha:1.0f];
            [UIView commitAnimations];
            break;
        }
        case UIGestureRecognizerStateFailed:{
            break;
        }
        case UIGestureRecognizerStatePossible:{
            break;
        }
        default:{
            CGContextRef context = UIGraphicsGetCurrentContext();
            [UIView beginAnimations:nil context:context]; //开始动画
            [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
            [UIView setAnimationDuration:1.0];           //设置动画持续时间
            [[self.view viewWithTag:999] setAlpha:0.0f];
            [UIView commitAnimations];
            break;
        }
    }
}

```

【代码解析】

本实例关键功能是工具栏在滚动时的显示和隐藏。下面就是这个知识点的详细讲解。

在本实例中，要想实现工具栏的显示和隐藏，首先，需要使用 `setTag` 方法对工具栏的 `tag` 值进行设置，代码如下：

```
[myToolbar setTag:999];
```

在使用的时候，使用 `viewWithTag:` 方法获取相应的子视图，其语法形式如下：

```
- (UIView *)viewWithTag:(NSInteger)tag;
```

其中，`(NSInteger)tag` 表示 `tag` 值。在本实例中就使用了 `viewWithTag:` 方法对 `tag` 值为 999 的工具栏进行了显示和隐藏功能，代码如下：

```
[[self.view viewWithTag:999] setAlpha:1.0f];
```

其中，999 表示 `tag` 值。

实例 69 网页浏览器

【实例描述】

本实例实现的功能是自制的网页浏览器的功能。用户可以在自制的网页浏览器的文本框中输入网址，进行相应网站的浏览，还可以打开 iOS 自带的 Safari 进行网站的浏览。运行

效果如图 5.18 所示。

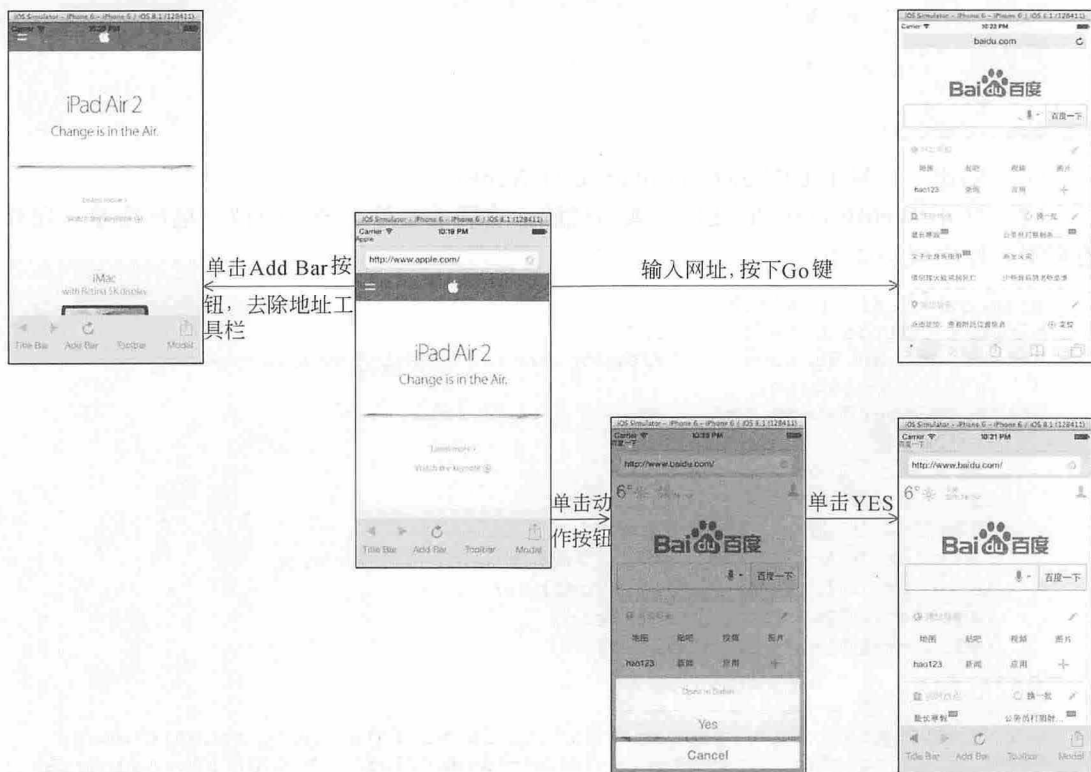


图 5.18 运行效果

【实现过程】

- (1) 创建一个项目，命名为“网页浏览器”。
- (2) 添加图像 1.png、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIToolbar 类的分类 Toolbar。
- (4) 打开 Toolbar.h 文件，编写代码，实现方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface UIToolbar (Toolbar)
- (void)replaceItem:(UIBarButtonItem *)oldItem withItem:(UIBarButtonItem *)item;
@end
```

- (5) 打开 Toolbar.m 文件，编写代码，进行声明方法的定义，实现 UIBarButtonItem 对象的替换功能。程序代码如下：

```
- (void)replaceItem:(UIBarButtonItem *)oldItem withItem:(UIBarButtonItem *)item {
    NSInteger buttonIndex = 0;
    //遍历
    for (UIBarButtonItem *button in self.items) {
        //判断 button 是否等于 oldItem
        if (button == oldItem) {
            NSMutableArray* newItems = [NSMutableArray arrayWithArray:
            self.items];
            [newItems replaceObjectAtIndex:buttonIndex withObject:item];
        }
    }
}
```



```

        self.items = newItem;
        break;
    }
    ++buttonIndex;
}
}

```

//替换对象

(6) 创建一个基于 UIViewController 类的 WebBrowser 类。

(7) 打开 WebBrowser.h 文件，编写代码，实现头文件、遵守协议、插座变量、属性的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "UIToolbar+Toolbar.h"
@interface WebBrowser : UIViewController<UIWebViewDelegate,
UITextFieldDelegate,
UIActionSheetDelegate>{
    //插座变量
    IBOutlet UIWebView *web;
    IBOutlet UIView *titleLabel;
    IBOutlet UILabel *titleLabel;
    IBOutlet UIActivityIndicatorView *loadingIndicator;
    IBOutlet UIToolbar *addressToolbar;
    IBOutlet UITextField *urlField;
    IBOutlet UIToolbar *toolbar;
}
//属性
@property (nonatomic, getter = isTitleBarVisible) BOOL showTitleBar;
@property (nonatomic, getter = isAddressBarVisible) BOOL showAddressBar;
.....
@property (nonatomic) BOOL _firstRequest;
@property (nonatomic, strong) NSString *_dataMimeType;
@end

```

(8) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 WebBrowser。这时新增的 View Controller 视图控制器就变为了 Web Browser 视图控制器。在 Identity 面板下，将 Storyboard ID 设置为 WebBrowser，选中 Use Storyboard ID 复选框。

(9) 对 Web Browser 视图控制器的设计界面进行设计，效果如图 5.19 所示。

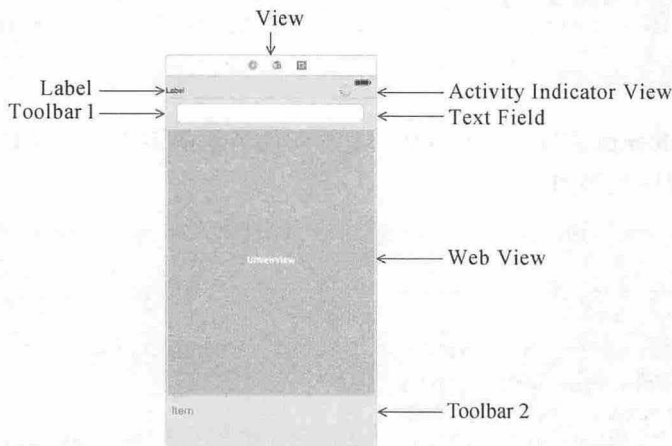


图 5.19 Web Browser 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 5-14 所示。

表 5-14 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	
View	Background: 浅灰色	与插座变量 titleToolbar 关联
Label	Font: System 11.0	与插座变量 titleLabel 关联
Activity Indicator View		与插座变量 loadingIndicator 关联
Toolbar1	Bar Tint: 浅灰色	与插座变量 addressToolbar 关联
Text Field		与插座变量 urlField 关联
Web View		与插座变量 web 关联
Toolbar2	Bar Tint: 浅灰色	与插座变量 toolbar 关联

(10) 打开 WebBrowser.m 文件, 编写代码, 实现网页浏览器的设置、网址输入以及加载等功能。使用的方法如表 5-15 所示。

表 5-15 WebBrowser.m文件中方法总结

方 法	功 能
initWithNibName:bundle:	初始化
viewDidLoad	视图加载后调用, 实现初始化
setUrl:	设置 url
setData:forMimeType:	设置数据
setHtml:	设置 html
setShowDoneButton:	设置 Done 按钮的显示
setShowTitleBar:	设置显示的标题栏
setShowTitleBar: animated:	设置显示的标题栏及动画
setShowAddressBar:	设置显示的地址工具栏
setShowAddressBar:animated:	设置显示的地址工具栏及动画
setShowToolBar:	设置显示的工具栏
setShowToolBar: animated:	设置显示的工具栏及动画
setVisibleComponentsAnimated:	设置显示的动画效果
setToolBarButtons	设置工具栏按钮
doneButtonPressed	按下 Done 按钮后的响应
actionButtonPressed	打开动作表单
backAction	返回
forwardAction	前进
refreshAction	加载
stopAction	停止
loadContent	加载内容
webViewDidStartLoad:	开始加载时调用
webViewDidFinishLoad:	加载完成
webView: didFailLoadWithError:	加载错误
textFieldShouldReturn:	按下 return 键
textFieldShouldClear:	执行 clear 按钮的操作
textFieldDidBeginEditing:	判断网页视图是否加载
textFieldDidEndEditing:	判断文本框是否为空
actionSheet:didDismissWithButtonIndex:	动作表单的响应

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。网页浏览器的设置需要使用 `initWithNibName:bundle:`、`viewDidLoad`、`setUrl:`、`setData:forMimeType:`、`setHtml:`、`setShowDoneButton:`、`setShowTitleBar:`、`setShowTitleBar:animated:`、`setShowAddressBar:`、`setShowAddressBar:animated:`、`setShowToolbar:`、`setShowToolbar:animated:`、`setVisibleComponentsAnimated:`、`setToolbarButtons`、`doneButtonPressed`、`actionButtonPressed`、`backAction`、`forwardAction`、`refreshAction`、`stopAction`、`loadContent`、`actionSheet:didDismissWithButtonIndex:`方法。其中，`initWithNibName:bundle:`方法实现了一些初始化的设置。程序代码如下：

```
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        showTitleBar = YES;           //显示标题栏
        showAddressBar = YES;         //显示地址栏
        showToolbar = YES;            //显示工具栏
        _firstRequest = YES;
    }
    return self;
}
```

`viewDidLoad` 方法在视图加载后调用，实现了一些初始值的设置。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    titleLabel.text = @"Loading..."; //设置文本内容
    web.delegate=self;
    web.scalesPageToFit=YES;         //自动缩放
    //对文本框的设置
    urlField.clearButtonMode = UITextFieldViewModeAlways;
    urlField.keyboardType = UIKeyboardTypeURL; //设置键盘类型
    urlField.returnKeyType = UIReturnKeyGo;
    urlField.delegate = self;         //设置委托
    [self setToolbarButtons];
    [self loadContent];
}
```

`setUrl:`方法实现对 url 网址的设置。程序代码如下：

```
- (void)setUrl:(NSURL *)inUrl {
    self->url = inUrl;
    self->data = nil;
    self->html = nil;
    if (self.view) {
        [self loadContent]; //加载内容
    }
}
```

`setVisibleComponentsAnimated:`方法实现当标题栏、地址栏、工具栏等可见时的动画效果的设置。程序代码如下：

```
- (void)setVisibleComponentsAnimated:(BOOL)animated {
```

```

CGRect titleFrame = titleToolbar.frame;           //获取标题栏的框架
CGRect addressFrame = addressToolbar.frame;        //获取地址栏的框架
CGRect toolbarFrame = toolbar.frame;               //获取工具栏的框架
CGRect webFrame = self.view.bounds;                //获取视图的边界
//判断是否显示标题栏
if (showTitleBar) {
    titleFrame.origin.y = 0;
    webFrame.origin.y += 34;
    webFrame.size.height -= 34;
} else {
    titleFrame.origin.y = 0 - titleFrame.size.height;
}
//判断是否显示地址栏
if (showAddressBar) {
    addressFrame.origin.y = (showTitleBar) ? 34 : 0;
    webFrame.origin.y += 44;
    webFrame.size.height -= 44;
} else {
    addressFrame.origin.y = 0 - addressFrame.size.height;
}
//判断是否显示工具栏
if (showToolbar) {
    webFrame.size.height -= 80;
}
toolbarFrame.origin.y = webFrame.origin.y + webFrame.size.height;
//判断是否有动画效果
if (animated) {
    //动画效果
    [UIView animateWithDuration:0.3
        animations:^(
            web.frame = webFrame;           //设置网页视图的框架
            titleToolbar.frame = titleFrame;
            addressToolbar.frame = addressFrame;
            toolbar.frame = toolbarFrame;    //设置工具栏的框架
        )];
} else {
    web.frame = webFrame;
    titleToolbar.frame = titleFrame;
    addressToolbar.frame = addressFrame;    //设置地址栏的框架
    toolbar.frame = toolbarFrame;
}
}

```

setToolBarButtons 方法实现工具栏中按钮的设置。程序代码如下:

```

- (void)setToolBarButtons {
    //创建并设置 UIBarButtonItem 对象
    UIBarButtonItem *vSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace target:nil action:nil];
    UIBarButtonItem *fSpace = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemFixedSpace target:nil action:nil];
    fSpace.width = 28.0f;           //设置 fSpace 对象的宽度
    _backButton = [[UIBarButtonItem alloc] initWithImage:[UIImage imageNamed:@"1.png"] style:UIBarButtonItemStylePlain target:self action:@selector(backAction)];
    _forwardButton = [[UIBarButtonItem alloc] initWithImage:[UIImage imageNamed:@"2.png"] style:UIBarButtonItemStylePlain target:self action:@selector(forwardAction)];
    _actionButton = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:

```

```

UIBarButtonItemSystemItemAction target:self action:@selector
(actionButtonPressed)];
_refreshButton = [[UIBarButtonItem alloc] initWithBarButtonSystem
Item:UIBarButtonSystemItemRefresh target:self action:@selector
(refreshAction)];
_refreshButton.tag = 3; //设置_refreshButton 对象的 tag 值
_stopButton = [[UIBarButtonItem alloc]
initWithBarButtonSystemItem:UIBarButtonSystemItemStop target:self
action:@selector(stopAction)];
_stopButton.tag = 3; //设置_stopButton 对象的 tag 值
_doneButton = [[UIBarButtonItem alloc] initWithTitle:@"Done" style:
UIBarButtonItemStyleBordered target:self action:@selector
(doneButtonPressed)];
NSMutableArray *toolbarItems;
//添加
toolbarItems = [NSMutableArray arrayWithObjects: _backButton,fSpace,
_forwardButton,fSpace,_refreshButton,vSpace,_actionButton,nil];
//判断是否显示 Done 按钮
if (showDoneButton) {
    [toolbarItems addObject:fSpace];
    [toolbarItems addObject:_doneButton];
}
[toolbar setItems:toolbarItems];
}

```

loadContent 方法实现网页视图中加载的内容。程序代码如下：

```

- (void)loadContent {
if (url) {
    //判断 web 是否正在加载
    if ([web isLoading])
        [web stopLoading]; //停止加载
    [web loadRequest:[NSURLRequest requestWithURL:url]];
    urlField.text = [url absoluteString]; //设置文本内容
} else if (data) {
    //加载数据
    [web loadData:data MIMEType:_dataMIMEType textEncodingName:@"utf-8"
    baseURL:nil];
} else if (html) {
    NSString *wrapperHTML = @"<html><head><meta name = \"viewport\"
    content = \"width = device-width\"><link rel=\"stylesheet\"
    media=\"all\" href=\"WebView.css\" /></head><body>%@</body></html>";
    NSString *finalHtml;
    if ([html rangeOfString:@"<html>"].location == NSNotFound) {
        finalHtml = [NSString stringWithFormat:wrapperHTML, html];
        //创建 finalHtml 对象
    } else {
        finalHtml = html;
    }
    [web loadHTMLString:finalHtml baseURL:[NSBundle mainBundle]
    bundleURL]; //加载 HTML 代码
}
}

```

actionButtonPressed 方法实现单击_actionButton 按钮对象后，显示动作表单的功能。程序代码如下：

```

- (IBAction)actionButtonPressed {
    UIActionSheet *as = [[UIActionSheet alloc] initWithTitle:@"Open in

```



```

Safari" delegate:self cancelButtonTitle:@"Cancel" destructiveButtonTitle:
nil otherButtonTitles:@"Yes", nil]; //创建动作表单
//判断程序运行的当前设置是否为 UIInterfaceIdiomPhone
if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInter
faceIdiomPhone) {
    if (self.view.superview) {
        [as showInView:self.view.superview]; //显示动作表单
    } else {
        [as showInView:self.view]; //显示动作表情
    }
} else {
    [as showFromBarButtonItem:_actionButton animated:YES];
}
}
}

```

actionSheet:didDismissWithButtonIndex:方法实现动作表单的响应。程序代码如下:

```

- (void)actionSheet:(UIActionSheet *)actionSheet didDismissWithButtonIndex:
(NSInteger)buttonIndex {
    if (buttonIndex == 0 && ! [[[web request] URL] absoluteString]
    isEqualToString:@""]) {
        [[UIApplication sharedApplication] openURL:[web request] URL]];
        //打开 Safari
    }
}

```

backAction、forwardAction、refreshAction、stopAction 方法对网页视图的控制。程序代码如下:

```

//返回
- (void)backAction {
    [web goBack];
}
//前进
- (void)forwardAction {
    [web goForward];
}
//加载
- (void)refreshAction {
    [web reload];
}
//停止
- (void)stopAction {
    [web stopLoading];
}

```

使用文本框输入网址需要使用 textFieldShouldReturn、textFieldShouldClear、textFieldDidBeginEditing、textFieldDidEndEditing:方法。其中, textFieldShouldReturn:方法实现按下 return 键后, 将 URL 网址显示在文本框中, 并且退出键盘。程序代码如下:

```

- (BOOL)textFieldShouldReturn:(UITextField *)textField {
    NSString *urlString;
    //判断文本框中字符串的长度是否为 0
    if ([textField.text rangeOfString:@"://"].length == 0) {
        urlString = [NSString stringWithFormat:@"http://%@", textField.
        text];
    } else {
        urlString = textField.text; //设置文本内容
    }
}

```

```

    }
    self.url = [NSURL URLWithString:urlString];
    [textField resignFirstResponder];
    return NO;
}

```

加载网址需要使用 `webViewDidStartLoad:`、`webViewDidFinishLoad:`、`webView:didFailLoadWithError:` 方法。其中，`webViewDidStartLoad:` 方法在网址视图开始加载时调用，实现加载指示器的加载等功能。程序代码如下：

```

- (void)webViewDidStartLoad:(UIWebView *)webView {
    _backButton.enabled = web.canGoBack;
    _forwardButton.enabled = web.canGoForward;
    [toolbar replaceItem:_refreshButton withItem:_stopButton];

    [loadingIndicator startAnimating]; //替换条目 //开始加载
    loadingIndicator.hidden=NO;
    titleLabel.text = @"Loading...";
    urlField.text = [[webView.request URL] absoluteString];
}

```

`webViewDidFinishLoad:` 方法在网页视图结束加载后调用，实现加载指示器的停止加载等功能。程序代码如下：

```

- (void)webViewDidFinishLoad:(UIWebView *)webView {
    _backButton.enabled = web.canGoBack;
    _forwardButton.enabled = web.canGoForward;
    [toolbar replaceItem:_stopButton withItem:_refreshButton];
    [loadingIndicator stopAnimating]; //结束加载
    loadingIndicator.hidden=YES;
    titleLabel.text = [web stringByEvaluatingJavaScriptFromString:
@"document.title"];
    urlField.text = [[webView.request URL] absoluteString];
    _firstRequest = NO;
}

```

`webView:didFailLoadWithError:` 方法在网页视图加载失败后调用，实现景观视图的弹出以及结束网页视图的加载。程序代码如下：

```

- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error
{
    if ([error code] != -999) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
        message:@"Could not connect to server." delegate:nil cancelButtonTitle:
@"Ok" otherButtonTitles:nil, nil]; //创建警告视图
        [alert show];
    }
    [self webViewDidFinishLoad:webView];
}

```

(11) 打开 `ViewController.h` 文件，编写代码，实现头文件以及属性的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "WebBrowser.h"
@interface ViewController : UIViewController
@property (nonatomic, strong) WebBrowser *browser;
@end

```


(12) 打开 ViewController.m 文件, 编写代码, 实现设计界面的设置以及按钮的响应。使用的方法如表 5-16 所示。

表 5-16 ViewController.m 文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
showmodal	单击 Modal 按钮
titlebar	单击 titlebar 按钮
addressbar	单击 addressbar 按钮
toolbar	单击 toolbar 按钮

其中, viewDidLoad 方法实现初始化的功能, 对界面进行设置。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //创建并设置网页浏览器对象
    browser = [self.storyboard instantiateViewControllerWithIdentifier:
@"WebBrowser"];
    browser.url = [NSURL URLWithString:@"http://apple.com"];
    CGRect frame = self.view.bounds;
    browser.view.frame = frame;                      //设置框架
    [self.view addSubview:browser.view];
    //按钮对象的创建和设置
    UIButton *button;
    button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button setTitle:@"Title Bar" forState:UIControlStateNormal];
    //设置按钮的标题
    [button addTarget:self action:@selector(titlebar) forControlEvents:
UIControlEventTouchUpInside];
    button.frame = CGRectMake(0, self.view.frame.size.height - 50, 80, 40);
    [self.view addSubview:button];                      //添加按钮对象
    button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button setTitle:@"Add Bar" forState:UIControlStateNormal];
    [button addTarget:self action:@selector(addressbar) forControlEvents:
UIControlEventTouchUpInside];
    button.frame = CGRectMake(85, self.view.frame.size.height - 50, 75, 40);
    //设置框架
    [self.view addSubview:button];
    button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button setTitle:@"Toolbar" forState:UIControlStateNormal];
    //设置标题
    [button addTarget:self action:@selector(toolbar) forControlEvents:
UIControlEventTouchUpInside];
    button.frame = CGRectMake(165, self.view.frame.size.height - 50, 80, 40);
    [self.view addSubview:button];                      //添加按钮对象
    button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [button setTitle:@"Modal" forState:UIControlStateNormal];
    //设置按钮的标题
    [button addTarget:self action:@selector(showmodal) forControlEvents:
UIControlEventTouchUpInside];
    button.frame = CGRectMake(250, self.view.frame.size.height - 50, 70, 40);
    //设置框架
    [self.view addSubview:button];
}

```

showmodal 方法实现单击 modal 按钮后，显示新的一个网页浏览器。程序代码如下：

```
- (void)showmodal {
    WebBrowser *modalBrowser=[self.storyboard instantiateViewController
    WithIdentifier:@"WebBrowser"];           //创建 modalBrowser 对象
    modalBrowser.showDoneButton = YES;
    modalBrowser.url = [NSURL URLWithString:@"http://apple.com"];
    [self presentViewController:modalBrowser animated:NO completion:nil];
    //显示视图控制器对象
}
```

【代码解析】

本实例关键功能网页视图的加载阶段。下面就是这个知识点的详细讲解。

一般网页视图加载需要 3 个阶段，当开始加载时，需要调用 `UIWebViewDelegate` 的 `webViewDidStartLoad:`方法，其语法形式如下：

```
- (void)webViewDidStartLoad:(UIWebView *)webView;
```

其中，`(UIWebView *)webView` 表示网页视图对象。当加载结束后，需要调用 `UIWebViewDelegate` 的 `webViewDidFinishLoad:`方法，其语法形式如下：

```
- (void)webViewDidFinishLoad:(UIWebView *)webView;
```

其中，`(UIWebView *)webView` 表示网页视图对象。当由于某些原因，长时间处于加载过程中时，加载就失败了，此时需要调用 `UIWebViewDelegate` 的方法，其语法形式如下：

```
- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error ;
```

其中，`(UIWebView *)webView` 表示网页视图对象；`(NSError *)error` 表示发生在加载过程中的错误。在本实例中就是使用了 `webViewDidStartLoad:`方法表示网页视图的开始加载，并且出现指示器进行加载。程序代码如下：

```
- (void)webViewDidStartLoad:(UIWebView *)webView {
    _backButton.enabled = web.canGoBack;
    _forwardButton.enabled = web.canGoForward;
    [toolbar replaceItem: refreshButton withItem:_stopButton];
    [loadingIndicator startAnimating];
    loadingIndicator.hidden=NO;
    titleLabel.text = @"Loading...";
    urlField.text = [[webView.request URL] absoluteString];
}
```

使用了 `webViewDidFinishLoad:`方法表示网页视图加载结束，并且在标签中显示了此时网页视图加载出的网站。程序代码如下：

```
- (void)webViewDidFinishLoad:(UIWebView *)webView {
    _backButton.enabled = web.canGoBack;
    _forwardButton.enabled = web.canGoForward;
    [toolbar replaceItem: stopButton withItem:_refreshButton];
    [loadingIndicator stopAnimating];
    loadingIndicator.hidden=YES;
    titleLabel.text = [web stringByEvaluatingJavaScriptFromString:
    @"document.title"];
    urlField.text = [[webView.request URL] absoluteString];
    _firstRequest = NO;
}
```

使用了 `webView:didFailLoadWithError:` 方法实现当网页视图长时间处于加载过程中时, 会出现警告视图并且停止加载。程序代码如下:

```
- (void)webView:(UIWebView *)webView didFailLoadWithError:(NSError *)error
{
    if ([error code] != -999) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Error"
        message:@"Could not connect to server." delegate:nil
        cancelButtonTitle:@"Ok" otherButtonTitles:nil, nil];
        [alert show];
    }
    [self webViewDidFinishLoad:webView];
}
```

第6章 地图

地图导航应用几乎是每一部手机中必备的应用软件。在地图中可以进行很多的操作，例如通过 GPS 获取用户当前位置，指定某一经纬度的位置，添加标注等。本实例主要讲解一下与地图相关的应用实例。

实例 70 地图切换器

【实例描述】

本实例的功能是实现 3 种地图类型之间的相互切换。当使用选择器选择其中的一个地图类型后，就会显示相应的地图类型。运行效果如图 6.1 所示。

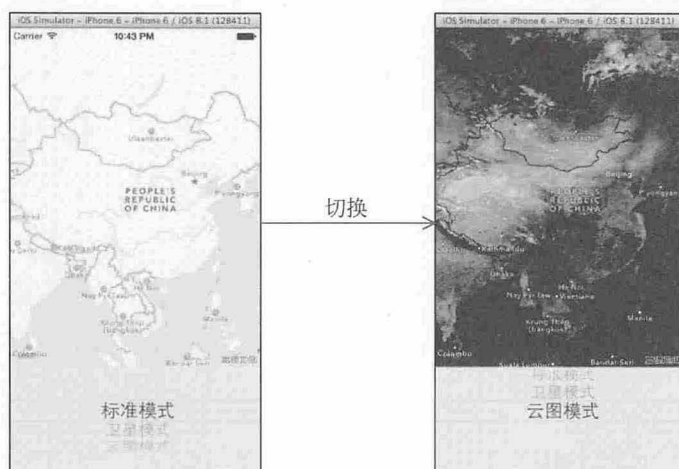


图 6.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“地图切换器”。
 - (2) 添加 MapKit.framework 到创建的项目中。
 - (3) 打开 ViewController.h 文件，编写代码，实现头文件、对象以及插座变量的声明。
- 程序代码如下：

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
@interface ViewController : UIViewController{
    NSArray *array;
    IBOutlet MKMapView *mapView;
}
@end
```

//声明关于地图的插座变量

(4) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 6.2 所示。

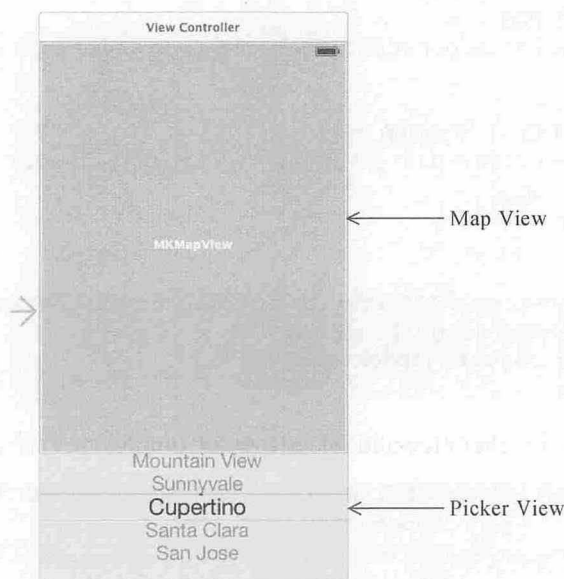


图 6.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-1 所示。

表 6-1 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 浅灰色	
Map View		与插座变量 mapView 关联
Picker View		

(5) 打开 ViewController.m 文件, 编写代码, 实现自定义选择器的内容填充以及地图的切换。使用的方法如表 6-2 所示。

表 6-2 ViewController.m 文件中方法总结

方法	功能
viewDidLoad	视图加载后调用, 实现初始化
numberOfComponentsInPickerView:	获取块数
pickerView:numberOfRowsInComponent:	获取行数
pickerView:titleForRow:forComponent:	获取内容
pickerView:didSelectRow:inComponent:	实现选择器的响应

自定义选择器的内容填充需要使用 viewDidLoad、numberOfComponentsInPickerView:、pickerView:numberOfRowsInComponent:、pickerView:titleForRow:forComponent:方法。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
```



```

// Do any additional setup after loading the view, typically from a nib.
array=[[NSArray alloc] initWithObjects:@"标准模式",@"卫星模式",@"云图模式",nil];
//创建数组
}
//设置自定义选择器的列数
-(NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView{
    return 1;
}
//设置自定义选择器在每一栏中返回的行数
-(NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:
(NSInteger) component{
    return [array count];
}
//填充内容
-(NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)
row forComponent:(NSInteger) component {
    return [array objectAtIndex:row];
}

```

地图的切换需要使用 pickerView:didSelectRow:inComponent:方法。程序代码如下：

```

-(void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
inComponent:(NSInteger) component{
    int i=[pickerView selectedRowInComponent:0];    //获取选中行数的索引值
    if(i==0){
        mapView.mapType=MKMapTypeStandard;        //设置地图的类型
    }else if (i==1){
        mapView.mapType=MKMapTypeSatellite;        //设置地图的类型
    }else{
        mapView.mapType=MKMapTypeHybrid;           //设置地图的类型
    }
}

```

【代码解析】

本实例关键功能是地图的类型改变。下面就是这个知识点的详细讲解。

地图的类型是通过 MKMapView 的 mapType 属性指定的，其语法形式如下：

```
@property(n nonatomic) MKMapType mapType;
```

其中，设置的属性值就是地图的类型，地图类型有 3 种，如表 6-3 所示。

表 6-3 地图类型

类 型	功 能
MKMapTypeStandard	标准地图模式
MKMapTypeSatellite	卫星地图模式
MKMapTypeHybrid	具有街道等信息的卫星地图模式（即云图模式）

实例 71 温度带换算器

【实例描述】

本实例的功能是利用经度实现一个温度带换算器的功能。当在文本框中输入纬度后，按下键盘上的 return 键，就会在圆形的地图上显示相应的温度带。其中，用于用户输入的

是文本框。运行效果如图 6.3 所示。

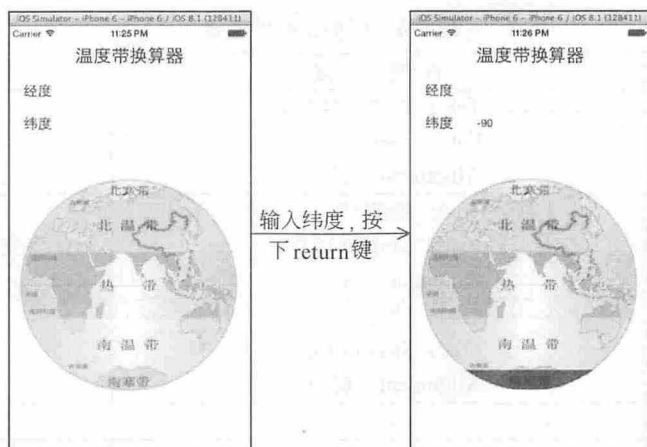


图 6.3 运行效果

【实现过程】

- (1) 创建一个项目，命名为“温度带换算器”。
- (2) 添加 CoreLocation.framework 到创建的项目中。
- (3) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量以及动作的声明。

程序代码如下：

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
@interface ViewController : UIViewController{
    IBOutlet UITextField *tf;                //声明关于文本框的插座变量
    IBOutlet UILabel *label;                //声明关于标签的插座变量
}
- (IBAction)hide: (id) sender;
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 6.4 所示。

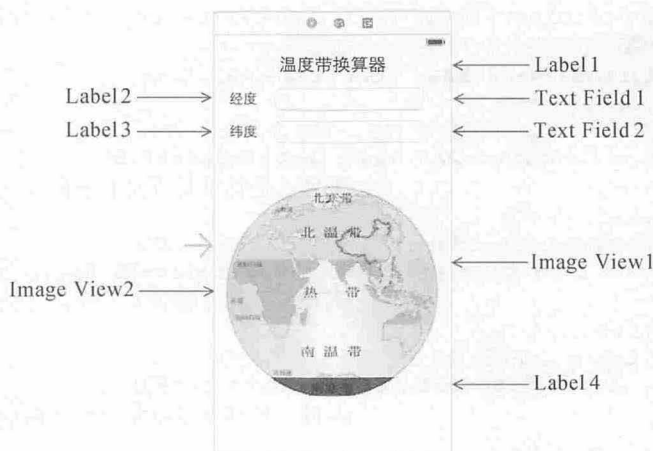


图 6.4 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-4 所示。

表 6-4 视图、控件设置

视图、控件	属性设置	其他
Label1	Text: 温度带换算器 Font: System 24.0 Alignment: 居中	
Label2	Text: 经度 Font: System 19.0 Alignment: 居中	与插座变量 mapView 关联
Label3	Text: 纬度 Font: System 19.0 Alignment: 居中	
Text Field1		
Text Field2		与插座变量 tf 关联 与动作 hide:关联 Did End Exit 与动作 hide:关联
Image View1	Image: 1.png	
Label4	Text: (空) Alpha: 0.6 Background: 黑色	与插座变量 label 关联
Image View2	Image: 2.png	

(5) 打开 ViewController.m 文件, 编写代码, 实现输入经纬度后显示选择相应的温带 (主要的是输入纬度, 经度可以不输)。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    label.hidden=YES;
}

- (IBAction)hide:(id)sender {
    [tf resignFirstResponder]; //关闭键盘
    NSString *l=[NSString stringWithFormat:@"%d",tf.text];
    CLLocationCoordinate2D a={0,[l floatValue]}; //设置位置坐标
    //判断 a 的纬度是否大于等于-23.5 小于等于 23.5
    if (a.longitude>=-23.5&&a.longitude<=23.5) {
        label.hidden=NO;
        label.frame=CGRectMake(15, 303, 290, 87);
    }else if (a.longitude>23.5&&a.longitude<=66.5) {
        //判断 a 的纬度是否大于等于-23.5 小于等于 23.5
        label.hidden=NO;
        label.frame=CGRectMake(15, 229, 290, 73);
    }else if (a.longitude>=-66.5&&a.longitude<-23.5) {
        //判断 a 的纬度是否大于等于-66.5 小于等于-23.5
        label.hidden=NO;
        label.frame=CGRectMake(15, 390, 290, 73);
    }else if (a.longitude>66.5&&a.longitude<=90) {
        //判断 a 的纬度是否大于等于 66.5 小于等于 90
        label.hidden=NO;
        label.frame=CGRectMake(15, 205, 290, 25);
    }else if (a.longitude>=-90&&a.longitude<-66.5) {

```

```

//判断 a 的纬度是否大于等于-90 小于等于-66.5
label.hidden=NO;
label.frame=CGRectMake(15, 463, 290, 25);
}
}

```

【代码解析】

本实例关键功能是纬度的获取。下面就是这个知识点的详细讲解。

经纬度的获取需要使用 `CLLocationCoordinate2D` 中的 `longitude` 字段和 `latitude` 字段。在本实例中使用了 `longitude` 字段获取了 `CLLocationCoordinate2D` 中对应的经度，并进行判断的。代码如下：

```

if (a.longitude>=-23.5&&a.longitude<=23.5) {
    label.hidden=NO;
    label.frame=CGRectMake(19, 303, 280, 87);
}

```

实例 72 地图导航

【实例描述】

本实例实现的是地图导航的功能。当用户在文本框中输入经纬度后，按下键盘上的按钮，就会在标签中显示输入的经纬度所指定的位置，当用户在分段控件中选择某一方式后，就会出现相应的导航路线。运行效果如图 6.5 所示。



图 6.5 运行效果

【实现过程】

- (1) 创建一个项目，命名为“地图导航”。
- (2) 打开 `ViewController.h` 文件，编写代码，实现插座变量、实例变量以及动作的声

明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet UIWebView *webView;           //声明关于网页视图的插座变量
    IBOutlet UILabel *Label1;             //声明关于标签的插座变量
    IBOutlet UITextField *tf1;
    IBOutlet UITextField *tf2;
    IBOutlet UILabel *label2;
    IBOutlet UITextField *tf3;           //声明关于文本框的插座变量
    IBOutlet UITextField *tf4;
    IBOutlet UISegmentedControl *segmented; //声明关于分段控件的插座变量
    //实例变量
    float starLat;
    float starLng;
    float endLat;
    float endLng;
}
//动作
- (IBAction)show:(id)sender;
- (IBAction)hide:(id)sender;
@end
```

（3）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 6.6 所示。

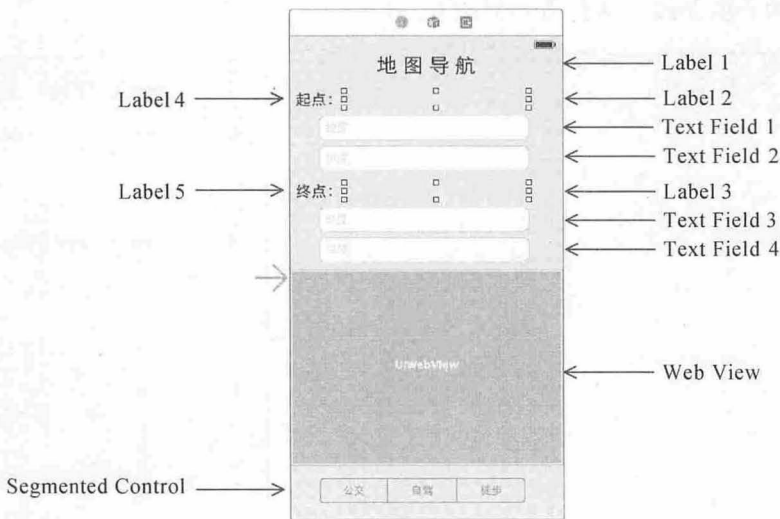


图 6.6 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-5 所示。

表 6-5 视图、控件设置

视图、控件	属性设置	其他
Label1	Text: 地图导航 Font: System 24.0 Alignment: 居中	

续表

视图、控件	属性设置	其他
Label2	Text: (空) Font: System 13.0	与插座变量 label1 关联
Label3	Text: (空) Font: System 13.0	与插座变量 label2 关联
Label4	Text: 起点 Font: System 18.0 Alignment: 居中	
Label5	Text: 终点 Font: System 18.0 Alignment: 居中	
Text Field1	Placeholder: 经度	与插座变量 tf1 关联 与动作 hide:关联 Did End Exit 与动作 hide:关联
Text Field2	Placeholder: 纬度	与插座变量 tf2 关联 与动作 hide:关联 Did End Exit 与动作 hide:关联
Text Field3	Placeholder: 经度	与插座变量 tf3 关联 与动作 hide:关联 Did End Exit 与动作 hide:关联
Text Field4	Placeholder: 纬度	与插座变量 tf4 关联 与动作 hide:关联 Did End Exit 与动作 hide:关联
Web View		与插座变量 webView 关联
Segmented Control	Segments: 3 Segment: Segment 0 Title: 公交 Segment: Segment 1 Title: 自驾 Segment: Segment 0 Title: 徒步	与插座变量 segmented 关联 与动作 show:关联

(4) 打开 ViewController.m 文件, 编写代码, 实现导航功能。使用的方法如表 6-6 所示。

表 6-6 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
GetHtmlFromUrl:	获取 html 数据
show:	实现导航
hide:	获取数据

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, GetHtmlFromUrl: 方法是获取 HTML 数据, 程序代码如下:

```
-(NSString*)GetHtmlFromUrl:(NSString *)urlStr
{
```

```

    ///获取 html 数据
    return [[NSString alloc] initWithContentsOfURL:[NSURL URLWithString:
urlStr] encoding:NSUTF8StringEncoding error:nil];
}

```

show:方法实现选择分段控件的某一段后会实现相应的导航。程序代码如下:

```

- (IBAction)show:(id)sender {
    NSInteger select=segmented.selectedSegmentIndex;//获取选择段的索引值
    if (select==0) {                                     //判断 select 是否等于 0
        //公交
        NSString *travelType = @"transit";
        NSString *urlStr = [NSString stringWithFormat:@"http://api.zdoz.net/daohang.aspx?TravelType=%@&startLat=%f&startLng=%f&endLat=%f&endLng=%f", travelType, starLat, starLng, endLat, endLng];
        [webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:urlStr]]]; //加载请求
    }else if (select==1){                                //判断 select 是否等于 1
        //自驾
        NSString *travelType = @"driving";
        NSString *urlStr = [NSString stringWithFormat:@"http://api.zdoz.net/daohang.aspx?TravelType=%@&startLat=%f&startLng=%f&endLat=%f&endLng=%f", travelType, starLat, starLng, endLat, endLng];
        //加载请求
        [webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:urlStr]]];
    }else {
        //徒步
        NSString *travelType = @"walking";
        NSString *urlStr = [NSString stringWithFormat:@"http://api.zdoz.net/daohang.aspx?TravelType=%@&startLat=%f&startLng=%f&endLat=%f&endLng=%f", travelType, starLat, starLng, endLat, endLng];
        [webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:urlStr]]]; //加载请求
    }
}

```

hide:方法实现通过输入经纬度获取开始位置以及结束位置，并输出。程序代码如下:

```

- (IBAction)hide:(id)sender {
    //关闭键盘
    [tf1 resignFirstResponder];
    [tf2 resignFirstResponder];
    [tf3 resignFirstResponder];
    [tf4 resignFirstResponder];
    //判断 4 个文本框的文本内容是否为空
    if (tf1.text.length!=0&&tf2.text.length!=0&& tf3.text.length!=0&& tf4.text.length!=0 ) {
        NSString *start1=[NSString stringWithFormat:@"%@",tf1.text];
        NSString *start2=[NSString stringWithFormat:@"%@",tf2.text];
        starLat=[start1 floatValue]; //将字符串转换为浮点类型
        starLng=[start2 floatValue];
        //获取 html 数据保存在字符串对象 startName 中
        NSString *startName = [self GetHtmlFromUrl:[NSString stringWithFormat:@"http://api.zdoz.net/geo2loc_2.aspx?lat=%f&lng=%f",starLat,starLng]];
        NSLog(@"%@",startName); //输出
        NSString *aa=[NSString stringWithFormat:@"http://api.zdoz.net/geo2loc_2.aspx?lat=%f&lng=%f",starLat,starLng];
    }
}

```

```

        NSLog(@"%@",aa);
        NSString *bb=[self GetHtmlFromUrl:aa];
        //获取 html 数据保存在字符串对象 startName 中

        NSLog(@"%@",bb);
        NSString *end1=[NSString stringWithFormat:@"%@",tf3.text];
        NSString *end2=[NSString stringWithFormat:@"%@",tf4.text];
        endLat=[end1 floatValue]; //将字符串转换为浮点类型
        endLng=[end2 floatValue];
        NSString *endName=[self GetHtmlFromUrl:[NSString stringWithFormat:
        @"http://api.zdoz.net/geo2loc_2.aspx?lat=%f1&lng=%f",endLat,endLng]];
        Label1.text = [NSString stringWithFormat:@"%@",startName];
        label2.text = [NSString stringWithFormat:@"%@",endName];
    }
}

```

【代码解析】

本实例关键功能是 HTML 数据的获取。下面就是这个知识点的详细讲解。NSString 的 initWithContentsOfURL:encoding:error:方法实现返回一个初始化的 NSString 对象,此对象是从一个给定的 URL 使用特定编码读取数据的字符串。其语法形式如下:

```

- (instancetype)initWithContentsOfURL:(NSURL *)url encoding:(NSStringEncoding)
enc error:(NSError **)error;

```

其中,(NSURL *)url 表示 URL;(NSStringEncoding)enc 表示字符编码;(NSError **)error 表示错误,如果有错误就返回错误信息。在本实例中就使用了 NSString 的 initWithContentsOfURL:encoding:error:方法获取了 URL 中读取的 HTML 的数据。代码如下:

```

return [[NSString alloc] initWithContentsOfURL:[NSURL URLWithString:urlStr]
encoding:NSUTF8StringEncoding error:nil];

```

其中,[NSURL URLWithString:urlStr]表示 URL; NSUTF8StringEncoding 表示字符编码; nil 表示错误,如果有错误也不返回错误信息。

实例 73 位置跟踪器

【实例描述】

本实例利用地图中当前位置的显示实现一个位置跟踪器的功能。在每 3 秒后实现一次更新功能。运行效果如图 6.7 所示。

【实现过程】

- (1) 创建一个项目,命名为“位置跟踪器”。
- (2) 添加 CoreLocation.framework 与 MapKit.framework 到创建的项目中。
- (3) 打开 ViewController.h 文件,编写代码,实现头文件、遵守协议以及插座变量的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import <CoreLocation/CoreLocation.h>
@interface ViewController : UIViewController<CLLocationManagerDelegate>{
    IBOutlet MKMapView *mapView; //声明关于地图视图的插座变量
}
@end

```



图 6.7 运行效果

(4) 打开 Main.storyboard 文件，从视图库中拖动 Map View 地图视图到设计界面中，并将声明的插座变量与此地图视图关联。

(5) 打开 ViewController.m 文件，编写代码，实现位置跟踪的功能。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    mapView.showsUserLocation=YES; //显示当前位置
    CLLocationManager *locationManager = [[CLLocationManager alloc] init]; //创建位置管理器
    locationManager.delegate=self; //设置代理
    locationManager.desiredAccuracy=kCLLocationAccuracyBest; //指定需要的精度级别
    locationManager.distanceFilter=1000.0f; //设置距离筛选器
    [locationManager startUpdatingLocation]; //启动位置管理器
    MKCoordinateSpan theSpan;
    //地图的范围 越小越精确
    theSpan.latitudeDelta=0.05;
    theSpan.longitudeDelta=0.05;
    MKCoordinateRegion theRegion;
    theRegion.center=[[locationManager location] coordinate]; //设置中心位置
    theRegion.span=theSpan;
    mapView.region=theRegion;
    [self performSelector:@selector(viewDidLoad) withObject:nil afterDelay:3]; //经过 3 秒后执行 viewDidLoad
}
```

【代码解析】

本实例关键功能是当前位置的显示、地图显示区域的设置以及当前位置的更新。下面依次讲解这 3 个知识点。

1. 当前位置的显示

当前位置的显示，需要使用 MKMapView 的 showsUserLocation 属性进行设置。其语法形式如下：

```
@property(n nonatomic) BOOL showsUserLocation;
```

其中，如果将该属性设置为 YES，表示在地图上显示用户的当前位置；如果将该属性设置为 NO，表示在地图上不显示用户的当前位置。在本实例中，就是将 MKMapView 的 showsUserLocation 属性设置为了 YES，才显示了用户的当前位置。代码如下：

```
mapView.showsUserLocation=YES;
```

2. 地图显示区域的设置

指定地图上的显示区域，需要使用 MKMapView 的 region 属性，其语法形式如下：

```
@property(n nonatomic) MKCoordinateRegion region;
```

在本实例中，就是使用了 MKMapView 的 region 属性对进行的显示区域进行了设置，代码如下：

```
mapView.region=theRegion;
```

3. 当前位置的更新

在本实例中，对于当前位置的不断更新使用的是 NSObject 的 performSelector:withObject:afterDelay:方法，它的功能是在当前线程中执行指定的方法，代码如下：

```
[self performSelector:@selector(viewDidLoad) withObject:nil afterDelay:0];
```

此代码的功能实现在 0 秒后执行 viewDidLoad 方法。

实例 74 指南针

【实例描述】

本实例实现的是指南针的功能。当用户旋转手机后，指南针始终执行磁北极的方向。运行效果如图 6.8 所示。

【实现过程】

- (1) 创建一个项目，命名为“指南针”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 CoreLocation.framework 到创建的项目中。
- (4) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、插座变量以及对象的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
@interface ViewController : UIViewController<CLLocationManagerDelegate>{
```



```
IBOutlet UIImageView *imageView; //声明插座变量
CLLocationManager *manager;
}
@end
```

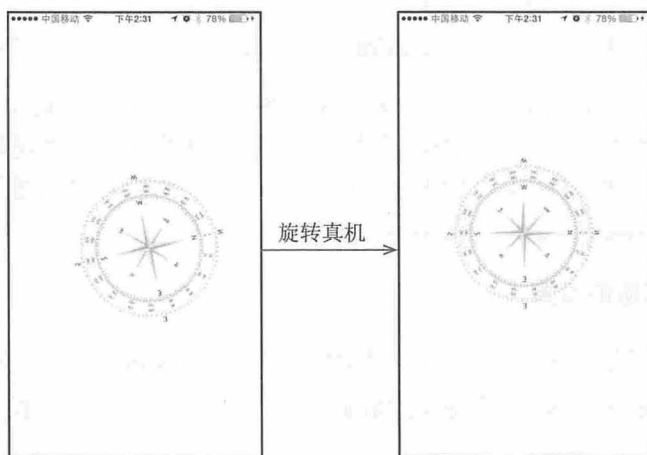


图 6.8 运行效果

(5) 打开 Main.storyboard 文件，从视图库中拖动 Image View 图像视图到设计界面中，将此视图的 Image 属性设置为 1.png，并且将此视图和声明的插座变量 imageView 关联。

(6) 打开 ViewController.m 文件，编写代码，实现指南针的功能。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    manager = [[CLLocationManager alloc] init];
    manager.delegate = self; //设置委托
    manager.desiredAccuracy = kCLLocationAccuracyBest; //指定需要的精度级别为最佳精度
    [manager startMonitoringSignificantLocationChanges]; //开发 Significant-Change 位置服务
    [manager startUpdatingHeading];
}
//更新指南针的方向时调用，实现指南针的旋转
- (void)locationManager:(CLLocationManager *)manager didUpdateHeading:
(CLLocationHeading *)newHeading {
    CGFloat heading = -1.0f * M_PI * newHeading.magneticHeading / 180.0f; //返回指南针磁北的方向
    imageView.center=CGPointMake(160, 284);
    imageView.transform = CGAffineTransformMakeRotation(heading); //旋转
}
```

【代码解析】

本实例关键功能是方向的更新以及指定方向的磁极。下面依次讲解这两个知识点。

1. 方向的更新

CLLocationManagerDelegate 的 locationManager:didUpdateHeading:方法实现方向的更新功能，当方向发生改变时就会调用此方法，其语法形式如下：

```
- (void)locationManager:(CLLocationManager *)manager didUpdateHeading:
(CLHeading *)newHeading
```

其中, (CLLocationManager *)manager 表示 CLLocationManager 对象, (CLHeading *)newHeading 表示 CLHeading 对象, 是一个新的方向数据。在本实例就使用了 locationManager:didUpdateHeading:方法实现了方向的功能功能, 代码如下:

```
- (void)locationManager:(CLLocationManager *)manager didUpdateHeading:
(CLHeading *)newHeading {
    .....
}
```

2. 指定方向的磁极

CLHeading 的 magneticHeading 属性可以获取磁北极的相对位置, 其语法形式如下:

```
@property(readonly, nonatomic) CLLocationDirection magneticHeading ;
```

在本实例中就是获取了磁北极的指南针磁北的方向, 从而实现图像的旋转功能。代码如下:

```
CGFloat heading = -1.0f * M_PI * newHeading.magneticHeading / 180.0f;
//返回指南针磁北的方向
.....
imageView.transform = CGAffineTransformMakeRotation(heading);
```

实例 75 驴友历程

【实例描述】

本实例实现的功能是驴友历程的应用。当单击获取当前位置按钮后, 就会在地图中显示当前有用的位置; 当单击“添加标注”按钮后, 会在用户所在的当前位置添加标注, 在标注的注释视图中会显示当前的位置以及当前的时间。运行效果如图 6.9 所示。



图 6.9 运行效果

【实现过程】

- (1) 创建一个项目，命名为“驴友历程”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 CoreLocation.framework 与 MapKit.framework 到创建的项目中。
- (4) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、插座变量、实例变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
#import <MapKit/MapKit.h>
@interface ViewController : UIViewController<CLLocationManagerDelegate>{
    CLLocationManager *locationManager;
    IBOutlet MKMapView *map;           //声明关于地图视图的插座变量
    float Lat;
    float Lng;
}
//动作
- (IBAction)add:(id)sender;
- (IBAction)biaozhu:(id)sender;
@end
```

- (5) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 6.10 所示。

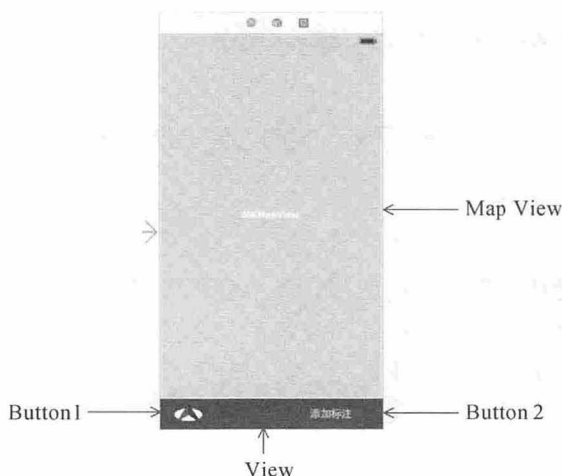


图 6.10 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-7 所示。

表 6-7 视图、控件设置

视图、控件	属性设置	其他
Map View		与插座变量 map 关联
Button1	Title: (空) Background: 1.png	与动作 add:关联
Button2	Title: 添加标注 Text Color: 白色	与动作 biaozhu:关联
View		

(6) 打开驴友历程-Info.plist 文件, 将 `CLLocationWhenInUseUsageDescription` 和 `CLLocationAlwaysUsageDescription` 添加到此文件中。

(7) 打开 `ViewController.m` 文件, 编写代码, 实现当前位置的获取以及添加标注等功能。程序代码如下:

```
//视图加载后调用, 实现初始化
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    map.showsUserLocation=YES;
    locationManager = [[CLLocationManager alloc] init];
    locationManager.delegate = self; //设置委托
    //判断是否实现了 requestWhenInUseAuthorization 方法
    if ([locationManager respondsToSelector:@selector(requestWhenInUse-
    Authorization)]) {
        [locationManager requestWhenInUseAuthorization];
    }
    [locationManager startUpdatingLocation]; //开启定位管理
}

//获取经纬度
- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:
(CLLocation *)newLocation fromLocation:(CLLocation *)oldLocation {
    [locationManager stopUpdatingLocation];
    Lat = newLocation.coordinate.latitude; //获取纬度
    Lng = newLocation.coordinate.longitude; //获取经度
}

//到达指定位置
- (IBAction)add:(id)sender {
    CLLocationCoordinate2D coor={Lat,Lng};
    [map setCenterCoordinate:coor animated:YES]; //设置中心点
}

//添加标注
- (IBAction)biaozhu:(id)sender {
    CLLocationCoordinate2D coor={Lat,Lng};
    MKPointAnnotation *ann=[[MKPointAnnotation alloc]init];
    //创建标准大头针对象
    ann.coordinate=coor;
    NSString *startName = [self GetHtmlFromUrl:[NSString stringWithFormat:
    @"http://www.zdoz.net/api/geo2loc.aspx?lat=%f&lng=%f",Lat,Lng]];
    //获取 HTML 数据
    ann.title=startName;
    NSString *str=[[NSString alloc]initWithFormat:@"我来过 时间:%@",[NSDate
    date]];
    ann.subtitle=str;
    [map addAnnotation:ann]; //添加标准大头针
    NSDictionary *address=[NSDictionary dictionaryWithObjectsAndKeys:startName,
    @"Country",startName,@"Locality", nil];
    //添加标记
    MKPlacemark *lun=[[MKPlacemark alloc]initWithCoordinate:coor
    addressDictionary: address];
    [map addAnnotation:lun]; //添加标记
}

- (NSString*)GetHtmlFromUrl:(NSString *)urlStr
{
    ///获取 html 数据
    return [[NSString alloc]initWithContentsOfURL:[NSURL URLWithString:
```

```
urlStr] encoding:NSUTF8StringEncoding error:nil];
}
```

【代码解析】

本实例关键功能是标记的添加以及标注的添加。下面依次讲解这两个知识点。

1. 标记的添加

在地图上做标记，是通过 MKPlacemark 类实现的。首先，需要对其进行初始化，通常使用 initWithCoordinate:addressDictionary:方法。其语法形式如下：

```
- (id)initWithCoordinate:(CLLocationCoordinate2D) coordinate addressDictionary:
(NSDictionary *)
addressDictionary ;
```

其中，(CLLocationCoordinate2D)coordinate 表示标记在地图上的坐标；(NSDictionary *)addressDictionary 表示一个字典对象。在本实例中就是使用了 initWithCoordinate:addressDictionary:方法实现了对标记的初始化功能。代码如下：

```
MKPlacemark *lun=[[MKPlacemark alloc]initWithCoordinate:coor
addressDictionary: address];
```

其中，coor 表示标记在地图上的坐标；address 表示一个字典对象。初始化之后，就可以将标记添加到地图上了，这时需要使用 MKMapView 类的 addAnnotation:方法实现，其语法形式如下：

```
- (void)addAnnotation:(id < MKAnnotation >)annotation;
```

在本实例中就是使用了 addAnnotation:方法将初始化后的标注添加到了地图上，代码如下：

```
[map addAnnotation:lun];
```

2. 标注的添加

在地图上添加一个标注大头针需要使用 MKPointAnnotation 类实现。使用此类，首先需要对其进行初始化，在本实例中使用的是 init 方法。其次，需要设置标注大头针在地图上的坐标点，需使用 MKPointAnnotation 的 coordinate 属性，其语法形式如下：

```
@property (nonatomic, assign) CLLocationCoordinate2D coordinate ;
```

在本实例中就使用了 coordinate 属性对标注大头针的坐标点进行了设置，代码如下：

```
ann.coordinate=coor;
```

在本实例中，标注的注释视图中还有标题以及副标题，它们分别使用了 MKAnnotation 的 title 属性和 subtitle 属性。其中，title 属性是对注释视图中标题的设置。其语法形式如下：

```
@property (nonatomic, readonly, copy) NSString *title;
```

在本实例中就使用了 title 属性对标注中注释视图的标题进行了设置，代码如下：

```
ann.title=startName;
```


subtitle 属性是对注释视图中副标题的设置。其语法形式如下：

```
@property (nonatomic, readonly, copy) NSString *subtitle;
```

在本实例中就使用了 subtitle 属性对标注中注释视图的副标题进行了设置，代码如下：

```
ann.subtitle=str;
```

都设置好以后，就可以使用 MKMapView 类的 addAnnotation:方法实现标注的添加了，在本实例中就使用了 MKMapView 类的 addAnnotation:方法实现了将设置好的标注添加到创建地图上，代码如下：

```
[map addAnnotation:lun];
```

实例 76 地图的位置查找

【实例描述】

本实例实现的是位置查找的功能。当用户在文本框中输入地址（必须是拼音或者是英文），按下键盘上的 return 键后，地图显示此地址的区域，并添加标记作为提醒用户的功能。运行效果如图 6.11 所示。



图 6.11 运行效果

【实现过程】

- (1) 创建一个项目，命名为“地图的位置查找”。
 - (2) 添加 CoreLocation.framework 与 MapKit.framework 到创建的项目中。
 - (3) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量以及动作的声明。
- 程序代码如下：

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
#import <CoreLocation/CoreLocation.h>
@interface ViewController : UIViewController{
```

```
IBOutlet UITextField *tf; //声明关于文本框的插座变量
IBOutlet MKMapView *map; //声明关于地图视图的插座变量
float Lat;
float Lng;
}
- (IBAction)search:(id)sender; //查找
@end
```

(4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面设计进行设计，效果如图 6.12 所示。

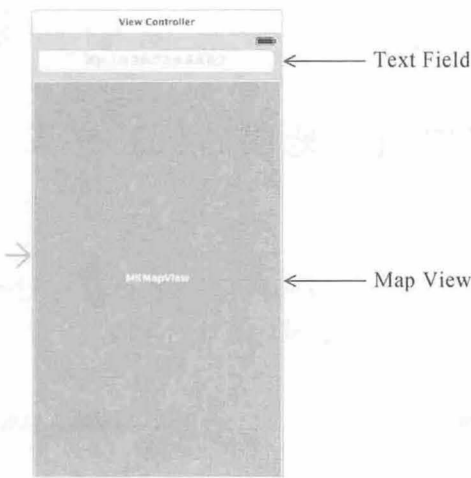


图 6.12 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-8 所示。

表 6-8 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	与插座变量 map 关联
Text Field	Placeholder: 请输入地名的汉语拼音或英文	与插座变量 tf 关联 与动作 search:关联 Did End Exit 与动作 search:关联
Map View		

(5) 打开 ViewController.m 文件，编写代码，实现位置的查找。程序代码如下：

```
- (IBAction)search:(id)sender {
//解析 Json 数据
NSString *url=[NSString stringWithFormat:@"http://maps.googleapis.com/
maps/api/geocode/json?address=%@&sensor=true",tf.text];
NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL
URLWithString:url]];
NSData *response = [NSURLConnection sendSynchronousRequest:request
returningResponse:nil error:nil];
//IOS5 自带解析类 NSJSONSerialization 从 response 中解析出数据放到字典中
NSDictionary *dic = [NSJSONSerialization JSONObjectWithData:response
options:NSJSONReadingMutableLeaves error:nil];
NSArray *results = [dic objectForKey:@"results"];
//获取字典中关键字为 results 的值
```

```

//遍历数组
for (NSDictionary * resultDetailDic in results)
{
    NSDictionary * locationDic=[[resultDetailDic objectForKey:@"geometry"]
    objectForKey:@"location"];
    NSString * lat=[locationDic objectForKey:@"lat"];
                                //获取字典中关键字为 lat 的值
    NSString * lng=[locationDic objectForKey:@"lng"];
                                //获取字典中关键字为 lng 的值

    Lat=[lat floatValue];
    Lng=[lng floatValue];
    NSLog(@"%f",Lat);
}

//获取 html 数据保存在 startName
NSString *startName = [self GetHtmlFromUrl:[NSString stringWithFormat:
@"http://api.zdoz.net/geo2loc_2.aspx?lat=%f&lng=%f",Lat,Lng]]
CLLocationCoordinate2D coor={Lat,Lng};
NSDictionary *address=[NSDictionary dictionaryWithObjectsAndKeys:
startName,@"Country",startName,@"Locality", nil];
//添加标记
MKPlacemark *lun=[[MKPlacemark alloc] initWithCoordinate:coor
addressDictionary: address];
[map addAnnotation:lun];
[map setCenterCoordinate:coor animated:YES];           //设置中心点
}
-(NSString*)GetHtmlFromUrl:(NSString *)urlStr
{
    ///获取 html 数据
    return [[NSString alloc] initWithContentsOfURL:[NSURL URLWithString:
urlStr] encoding:NSUTF8StringEncoding error:nil];
}

```

【代码解析】

本实例关键功能是地图显示区域的改变。下面就是这个知识点的详细讲解。

根据提供的经纬度为中心原点改变地图的显示区域，需要使用 MKMapView 的 setCenterCoordinate:animated:方法，其语法形式如下：

```

- (void)setCenterCoordinate:(CLLocationCoordinate2D)coordinate animated:
(BOOL)animated;

```

其中，(CLLocationCoordinate2D)coordinate 表示坐标点；(BOOL)animated 表示是否会出现动画效果，此项如果设置为 YES 表示有动画效果；如果设置为 NO，表示没有动画效果。在本实例中就使用了 setCenterCoordinate:animated:方法实现将地图的显示区域进行了改变，代码如下：

```

[map setCenterCoordinate:coor animated:YES];

```

其中，coor 表示坐标点；YES 表示有动画效果。

实例 77 3D 地图

【实例描述】

在 iOS 7 后，手机上就可以实现 3D 地图效果了。本实例就是实现了地图的 3D 显示效

果。运行效果如图 6.13 所示。

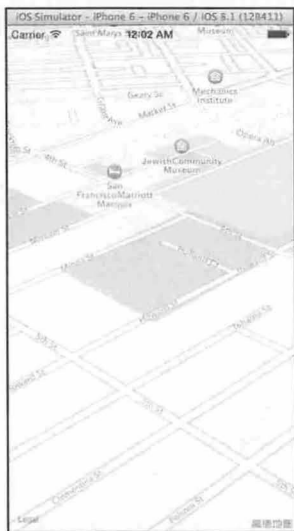


图 6.13 运行效果

【实现过程】

- (1) 创建一个项目，命名为“3D 地图”。
- (2) 添加 MapKit.framework 到创建的项目中。
- (3) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
@interface ViewController : UIViewController{
    IBOutlet MKMapView *mapView;
}
@end
```

- (4) 打开 Main.storyboard 文件，从视图库中拖动 Map View 地图视图到 View Controller 视图控制器的设计界面上，并将声明的 mapView 插座变量与此地图视图关联。

- (5) 打开 ViewController.m 文件，编写代码，实现 3D 地图的显示。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    mapView.centerCoordinate = CLLocationCoordinate2DMake(37.78275123,
        -122.40416442);
    mapView.camera.altitude = 200; //设置海拔
    mapView.camera.pitch = 70; //角度
    mapView.showsBuildings = YES;
}
```

【代码解析】

本实例关键功能是地图的 3D 效果。下面就是这个知识点的详细讲解。

在 iOS 7 中引入了一个新的 MKMapCamera 类，它将一个 MKMapCamera 对象添加到

地图上,在指明位置、角度和方向后,呈现 3D 的效果。对于角度、方法等的调整需要使用一些属性,如表 6-9 所示。

表 6-9 属性

方 法	功 能
centerCoordinate	地图视图的中心坐标
heading	方向
pitch	角度
altitude	海拔

在本实例中就是使用了 MKMapCamera 类实现了地图的 3D 效果,代码如下:

```
mapView.centerCoordinate = CLLocationCoordinate2DMake(37.78275123, -122.40416442);
mapView.camera.altitude = 200;
mapView.camera.pitch = 70;
mapView.showsBuildings = YES;
```

实例 78 旋转的地图

【实例描述】

本实例实现的功能是让 3D 地图进行旋转。当单击 Start 按钮后,进行旋转,并且按钮上的标题会改为 Stop;当单击此时的 Stop 按钮后旋转的按钮就会停止旋转,并且此时按钮的标题会变为 Start。运行效果如图 6.14 所示。



图 6.14 运行效果

【实现过程】

- (1) 创建一个项目,命名为“旋转的地图”。
- (2) 添加 MapKit.framework 到创建的项目中。
- (3) 创建一个基于 NSObject 类的 Rotating 类。
- (4) 打开 Rotating.h 文件,编写代码,实现头文件、宏定义、属性以及方法的声明。

程序代码如下:


```

#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>                                //头文件
//宏定义
#define DEFAULT_HEADING 0
#define DEFAULT_PITCH 45
#define DEFAULT_ALTITUDE 100
#define DEFAULT_HEADING_STEP 10
@interface Rotating : NSObject
//属性
@property (nonatomic, weak) MKMapView *mapView;
@property (nonatomic, assign) BOOL rotating;
@property (nonatomic, assign) double headingStep;
//方法
- (id)initWithMapView:(MKMapView *)mapView;
- (void)startRotating;
- (void)startRotatingWithCoordinate:(CLLocationCoordinate2D)coordinate;
//设置 MKMapCamera 的一些属性,从而进行旋转- (void)startRotatingWithCoordinate:
(CLLocationCoordinate2D)coordinate heading:(CLLocationDirection)heading
pitch:(CGFloat)pitch altitude:(CLLocationDistance)altitude headingStep:
(double)headingStep;
- (void)continueRotating;                                //继续旋转
- (void)stopRotating;
- (BOOL)isStopped;
@end

```

(5) 打开 Rotating.m 文件,编写代码,实现 MKMapCamera 对象即摄像头对象的旋转。使用的方法如表 6-10 所示。

表 6-10 Rotating.m文件中方法总结

方 法	功 能
initWithMapView:	初始化
startRotatingWithCoordinate: heading:pitch:altitude:headingStep:	设置 MKMapCamera 的一些属性,从而进行旋转
startRotatingWithCoordinate:	以坐标点为进行旋转
startRotating	开始旋转
stopRotating	结束旋转
isStopped	获取是否旋转
continueRotating	继续旋转

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, startRotatingWithCoordinate: heading:pitch:altitude:headingStep:方法实现对 MKMapCamera 的一些属性进行设置,从而实现旋转的功能。程序代码如下:

```

- (void)startRotatingWithCoordinate:(CLLocationCoordinate2D)coordinate
heading:(CLLocationDirection)heading pitch:(CGFloat)pitch altitude:
(CLLocationDistance)altitude headingStep:(double)headingStep {
    MKMapCamera *camera = [MKMapCamera new];
    camera.centerCoordinate = coordinate;    //设置读入的坐标
    camera.heading = heading;                //设置方向
    camera.pitch = pitch;                    //设置角度
    camera.altitude = altitude;              //设置海拔
    [self.mapView setCamera:camera animated:YES];
    self.rotating = YES;
    self.headingStep = headingStep;
}

```

startRotatingWithCoordinate:方法实现以坐标点为进行旋转的功能。程序代码如下:

```
- (void)startRotatingWithCoordinate:(CLLocationCoordinate2D)coordinate {
    [self startRotatingWithCoordinate:coordinate heading:DEFAULT_HEADING
    pitch:DEFAULT_PITCH altitude:DEFAULT_ALTITUDE headingStep:DEFAULT_
    HEADING_STEP];
}
```

continueRotating 方法实现继续进行旋转的功能。程序代码如下:

```
- (void) continueRotating {
    MKMapCamera *camera = [self.mapView.camera copy];
    camera.heading = fmod(camera.heading + self.headingStep, 360);
    //设置方向

    [UIView animateWithDuration:0.5 delay:0 options:UIViewAnimationOption-
    CurveLinear animations:^(
        self.mapView.camera = camera;
        //设置摄像头
    ) completion:nil];
}
```

(6) 打开 ViewController.h 文件,编写代码实现头文件、宏定义、插座变量、实例变量以及动作的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
//头文件
#import <MapKit/MapKit.h>
#import "Rotating.h"
#define DEFAULT_COORDINATE CLLocationCoordinate2DMake(37.78275123, -122.
40416442)
@interface ViewController : UIViewController<MKMapViewDelegate>{
    //插座变量
    IBOutlet MKMapView *map;
    IBOutlet UIButton *startStopRotating;
    Rotating *rotCamera;
    BOOL rotating;
}
- (IBAction)start:(id)sender;
@end
```

(7) 开发 Main.storyboard 文件,对 View Controller 视图控制器的设计界面进行设计,效果如图 6.15 所示。

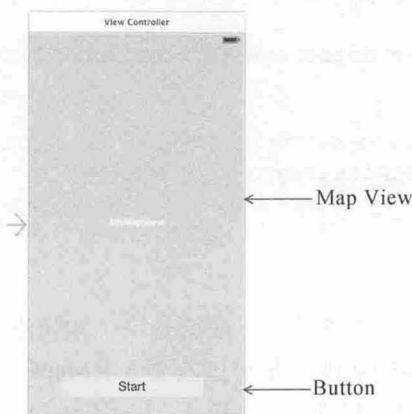


图 6.15 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-11 所示。

表 6-11 视图、控件设置

视图、控件	属 性 设 置	其 他
Button	Title: Start Font: System 19.0 Text Color: 黑色 Alpha: 0.8 Background: 前灰色	与插座变量 startStopRotating 关联 与动作 start:关联
Map View		与插座变量 map 关联

(8) 打开 ViewController.m 文件，编写代码，实现地图的旋转功能。程序代码如下：

```
//视图加载后调用，实现初始化
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [startStopRotating setTitle:@"Start" forState:UIControlStateNormal];
    //设置地图视图对象
    map.centerCoordinate = DEFAULT_COORDINATE;
    map.delegate = self; //设置委托
    map.showsBuildings = YES;
    //添加标注
    MKPointAnnotation *annotationPoint = [[MKPointAnnotation alloc] init];
    annotationPoint.coordinate = DEFAULT_COORDINATE;
    annotationPoint.title = @"Center"; //设置标题
    [map addAnnotation:annotationPoint];
    rotCamera = [[Rotating alloc] initWithMapView:map];
}

//判断摄像头是否停止旋转，如果没有执行继续旋转
- (void)mapView:(MKMapView *)mapView regionDidChangeAnimated:(BOOL)animated {
    if ([rotCamera isStopped] == NO) {
        [rotCamera continueRotating]; //调用 continueRotating 方法实现继续旋转
    }
}

//单击按钮
- (IBAction)start:(id)sender {
    if ([startStopRotating.titleLabel.text isEqualToString:@"Start"] == YES) {
        [startStopRotating setTitle:@"Stop" forState:UIControlStateNormal];
        //设置标题
        [rotCamera startRotatingWithCoordinate:DEFAULT_COORDINATE];
        //开始旋转
    } else {
        [startStopRotating setTitle:@"Start" forState:UIControlStateNormal];
        [rotCamera stopRotating]; //结束旋转
    }
}
```

【代码解析】

本实例关键功能是 3D 地图的旋转。下面就是这个知识点的详细讲解。

在本实例中 3D 地图旋转的实现，是通过改变 MKMapCamera 对象即摄像头对象的方向，从而实现旋转功能，使用的属性是 heading，其代码如下：

```
camera.heading = fmod(camera.heading + self.headingStep, 360);
```

此实例的程序执行流程图如图 6.16 所示。

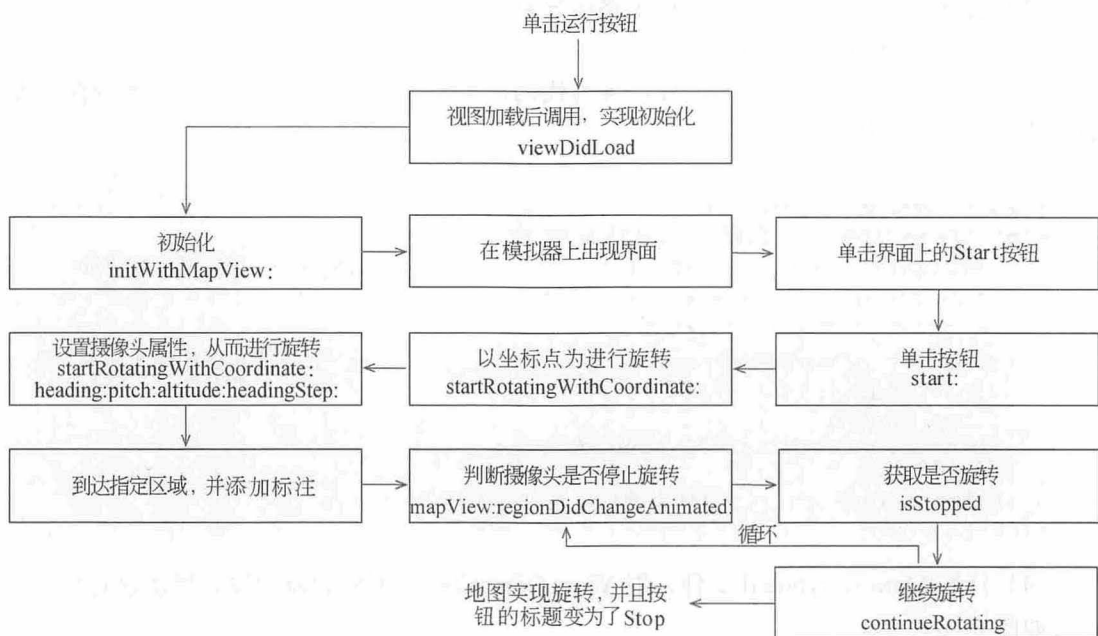


图 6.16 程序执行流程

实例 79 行车路线导航

【实例描述】

本实例实现的是行车路线导航功能。它可以获取从当前位置出发到达指定位置的路线。当用户在文本框中输入经纬度后, 单击“查找”按钮, 会根据经纬度显示出具体的位置信息, 并且还在地图上画出一条行车路线。运行效果如图 6.17 所示。



图 6.17 运行效果

【实现过程】

(1) 创建一个项目，命名为“行车路线导航”。

(2) 添加 MapKit.framework 到创建的项目中。

(3) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量、实例变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>
@interface ViewController : UIViewController{
    IBOutlet UILabel *toLabel;           //声明关于标签的插座变量
    IBOutlet UITextField *toLat;         //声明关于文本框的插座变量
    IBOutlet UITextField *toLng;
    IBOutlet MKMapView *map;             //声明关于地图视图的插座变量
    float endLat;
    float endLng;
    CLLocationCoordinate2D _coordinate;
}
- (IBAction)search:(id)sender;
@end
```

(4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 6.18 所示。

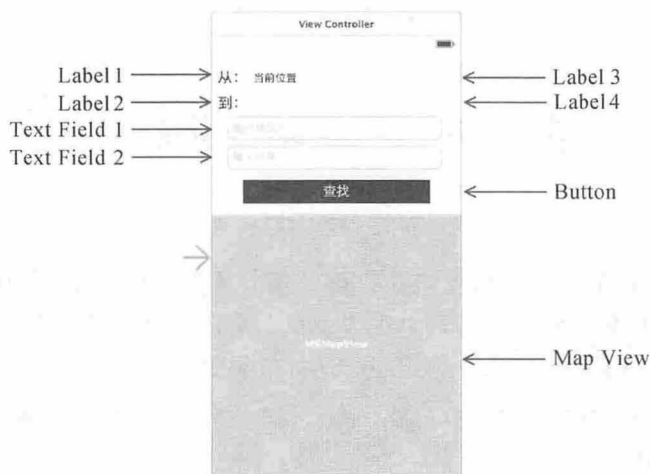


图 6.18 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-12 所示。

表 6-12 视图、控件设置

视图、控件	属性设置	其他
Label1	Text: 从: Font: System 19.0 Alignment: 居中	
Label2	Text: 到: Font: System 19.0 Alignment: 居中	

续表

视图、控件	属 性 设 置	其 他
Label3	Text: 当前位置 Font: System 14.0	
Label4	Text: (空) Font: System 14.0	与插座变量 toLabel 关联
Text Field1	Placeholder: 输入纬度	与插座变量 toLat 关联
Text Field2	Placeholder: 输入经度	与插座变量 toLng 关联
Button	Title: 查找 Font: System 17.0 Text Color: 白色 Background: 黑色	与动作 search:关联
Map View		与插座变量 map 关联 与 delegate 关联

(5) 打开 ViewController.m 文件, 编写代码, 实现导航功能。使用的方法如表 6-13 所示。

表 6-13 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
GetHtmlFromUrl:	获取 HTML 数据
search:	单击“查找”按钮
findDirectionsFrom:to:	获取路线
mapView: rendererForOverlay:	渲染
mapView:didUpdateUserLocation:	位置更新
setMapRegionWithCoordinate:	设置地图显示的区域

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, search:方法实现单击“查找”按钮后的响应, 即显示导航。程序代码如下:

```
- (IBAction)search:(id)sender {
    //关闭键盘
    [toLat resignFirstResponder];
    [toLng resignFirstResponder];
    //判断文本框 toLat 和 toLng 是否不为空
    if (toLat.text.length!=0 && toLng.text.length!=0 ) {
        NSString *end1=[NSString stringWithFormat:@"%@",toLat.text];
        NSString *end2=[NSString stringWithFormat:@"%@",toLng.text];
        endLat=[end1 floatValue];
        endLng=[end2 floatValue];
        //获取 HTML 数据保存到 endName 字符串对象中
        NSString *endName =[self GetHtmlFromUrl:[NSString stringWithFormat:
            @"http://api.zdoz.net/geo2loc_2.aspx?lat=%f&lng=%f",endLat,endLng]];
        toLabel.text = [NSString stringWithFormat:@"%@",endName];
        //设置标签中文本的内容

        //设置坐标
        CLLocationCoordinate2D fromCoordinate = _coordinate;
        CLLocationCoordinate2D toCoordinate = CLLocationCoordinate2DMake
            (endLat,endLng);
        //添加标记
        NSDictionary *address=[NSDictionary dictionaryWithObjectsAndKeys:
            endName,@"Country",endName,@"Locality", nil];
```

```

MKPlacemark *lun=[[MKPlacemark alloc] initWithCoordinate:toCoordinate
addressDictionary: address];
[map addAnnotation:lun];
//设置开始和结束的目的地
MKPlacemark *fromPlacemark = [[MKPlacemark alloc] initWithCoordinate:
fromCoordinate addressDictionary:nil]; //创建开始地的标记
MKPlacemark *toPlacemark = [[MKPlacemark alloc] initWithCoordinate:
toCoordinate addressDictionary:nil]; //创建结束地的标记
MKMapItem *fromItem = [[MKMapItem alloc] initWithPlacemark:
fromPlacemark];
MKMapItem *toItem = [[MKMapItem alloc] initWithPlacemark:
toPlacemark];
[self findDirectionsFrom:fromItem to:toItem]; //获取路线
}
}

```

findDirectionsFrom: to:方法实现对路线的获取。程序代码如下：

```

- (void)findDirectionsFrom: (MKMapItem *)source to: (MKMapItem *)destination
{
    MKDirectionsRequest *request = [[MKDirectionsRequest alloc] init];
    request.source = source;
    request.destination = destination; //设置路线的终点
    request.requestsAlternateRoutes = YES; //设置为获取所有的路线
    MKDirections *directions = [[MKDirections alloc] initWithRequest:request];
    //计算真实的路径
    [directions calculateDirectionsWithCompletionHandler:
    ^(MKDirectionsResponse *response, NSError *error) {
        //判断是否有错
        if (error) {
            NSLog(@"error:%@", error);
        }
        else {
            MKRoute *route = response.routes[0];
            [map addOverlay:route.polyline]; //添加覆盖对象
        }
    }];
}

```

mapView:rendererForOverlay:方法实现对路线的渲染功能。程序代码如下：

```

- (MKOverlayRenderer *)mapView: (MKMapView *)mapView rendererForOverlay:
(id<MKOverlay>)overlay
{
    MKPolylineRenderer *renderer = [[MKPolylineRenderer alloc]
initWithOverlay:overlay];
    renderer.lineWidth = 5.0; //设置线宽
    renderer.strokeColor = [UIColor purpleColor]; //设置线的颜色
    return renderer;
}

```

mapView: didUpdateUserLocation:方法实现了位置的更新。程序代码如下：

```

- (void)mapView: (MKMapView *)mapView didUpdateUserLocation: (MKUserLocation *)
userLocation
{
    _coordinate.latitude = userLocation.location.coordinate.latitude;
    //获取纬度
    _coordinate.longitude = userLocation.location.coordinate.longitude;
}

```

```

//获取经度
[self setMapRegionWithCoordinate:_coordinate];
}

```

setMapRegionWithCoordinate:方法实现了对地图显示的区域进行设置。程序代码如下:

```

- (void)setMapRegionWithCoordinate:(CLLocationCoordinate2D)coordinate
{
    MKCoordinateRegion region;
    region = MKCoordinateRegionMake(coordinate, MKCoordinateSpanMake
    (.1, .1));
    MKCoordinateRegion adjustedRegion = [map regionThatFits:region];
    [map setRegion:adjustedRegion animated:YES]; //设置区域
}

```

【代码解析】

本实例关键功能是获取路线信息以及渲染功能。下面依次讲解这两个知识点。

1. 获取路线信息

iOS 添加了一个新的类 MKDirections 类,通过此类可以从移动设备那里请求获得方向相关的线路信息。通过获得的线路信息可以创建一个图层并显示到地图中。在本实例中,就是使用了此类实现获取了从当前位置出发到达指定位置的路线信息。它的使用,首先需要创建并初始化一个 MKDirections 对象,这时使用 MKDirections 的 initWithRequest:方法,语法形式如下:

```

- (instancetype)initWithRequest:(MKDirectionsRequest *)request;

```

其中, (MKDirectionsRequest *)request 表示包含开始和结束点的路线请求对象。在本实例中就是使用了 initWithRequest:方法实现了 MKDirections 对象的创建以及初始化功能。程序代码如下:

```

MKDirections *directions = [[MKDirections alloc] initWithRequest:request];

```

其中, request 表示请求对象。创建并初始化对象后,就可以对真实的路线进行计算了,此时需要使用 MKDirections 的 calculateDirectionsWithCompletionHandler:方法实现,其语法形式如下:

```

- (void)calculateDirectionsWithCompletionHandler:(MKDirectionsHandler)
completionHandler;

```

其中, (MKDirectionsHandler)completionHandler 表示 completion handling 块。在本实例中,就是使用了 calculateDirectionsWithCompletionHandler:方法实现了对真实路线的计算。程序代码如下:

```

[directions calculateDirectionsWithCompletionHandler:^(MKDirectionsResponse
*response, NSError *error) {
    if (error) {
        NSLog(@"error:%@", error);
    } else {
        MKRoute *route = response.routes[0];
        [map addOverlay:route.polyline];
    }
}];

```

2. 渲染

在 iOS 7 中使用 MKPolylineRenderer 类实现渲染的功能。在本实例就使用了 MKPolylineRenderer 类实现了将计算出的路线进行渲染，并将其显示在地图上。代码如下：

```
MKPolylineRenderer *renderer = [[MKPolylineRenderer alloc] initWithOverlay:  
overlay];  
renderer.lineWidth = 5.0;  
renderer.strokeColor = [UIColor purpleColor];  
return renderer;
```

实例 80 时区换算器

【实例描述】

本实例实现的功能是一个时区换算器。当界面中的指示器指向某一位置时，就会在标签中显示此位置的时区。运行效果如图 6.19 所示。

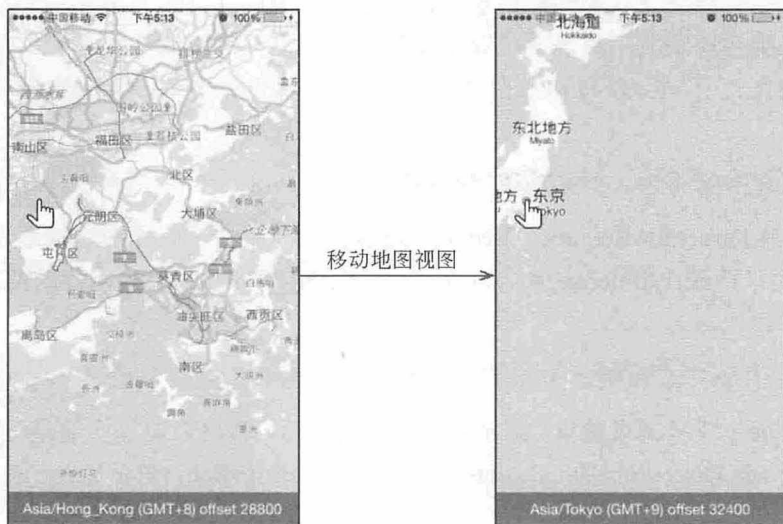


图 6.19 运行效果

【实现过程】

- (1) 创建一个项目，命名为“时区换算器”。
- (2) 添加 MapKit.framework 和 CoreLocation.framework 到创建的项目中。
- (3) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (4) 添加 timezonesDB.json 文件到创建项目的 Supporting Files 文件夹中。
- (5) 创建一个基于 CLLocation 类的分类 TimeZone 类。
- (6) 创建一个基于 NSObject 类的 TimeZone 类。
- (7) 打开 CLLocation+TimeZone.h 文件，编写代码，实现头文件以及方法的声明。程序代码如下：

```
#import <CoreLocation/CoreLocation.h>
```

```
#import "TimeZone.h"
@interface CLLocation (TimeZone)
- (NSTimeZone *)timeZone;           //获取时区
@end
```

(8) 打开 CLLocation+TimeZone.m 文件, 编写代码, 实现时区对象的获取。程序代码如下:

```
- (NSTimeZone *)timeZone {
    NSTimeZone *timeZone = [[TimeZone sharedInstance] timeZoneWithLocation:
self];           //创建时区对象
    return timeZone;
}
```

(9) 打开 TimeZone.h 文件, 编写代码, 实现头文件、属性以及方法的声明。程序代码如下:

```
#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>
#import "CLLocation+TimeZone.h"
@interface TimeZone : NSObject
@property (nonatomic, strong) NSArray *timeZonesDB;
- (NSTimeZone *)timeZoneWithLocation:(CLLocation *)location;
                                                    //通过位置获取时区
+ (TimeZone *)sharedInstance;           //获取单例
@end
```

(10) 打开 TimeZone.m 文件, 编写代码, 实现时区的获取。使用的方法如表 6-14 所示。

表 6-14 TimeZone.m 文件中方法总结

方 法	功 能
timeZoneWithLocation:	通过位置获取时区
timeZonesDB	获取 timeZonesDB.json 文件中解析的内容
importDataBaseFromFile:	从文件中解析内容
closesZoneInfoWithLocation: source:	获取最接近的距离
filteredTimeZonesWithCountyCode:	获取与 NSPredicate 对象匹配的内容
timeZoneWithDictionary:	获取时区
sharedInstance	获取单例

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, timeZoneWithLocation: 方法实现通过位置获取时区的功能。代码如下:

```
- (NSTimeZone *)timeZoneWithLocation:(CLLocation *)location {
    NSDictionary *closeseseZoneInfo = [self closeseseZoneInfoWithLocation:
location source:self.timeZonesDB];           //创建字典对象
    //判断 closeseseZoneInfo 对象是否为空
    if (closeseseZoneInfo == nil) {
        return [NSTimeZone systemTimeZone];           //获取系统时区
    }
    NSTimeZone *timeZone = [self timeZoneWithDictionary:closeseseZoneInfo];
    //判断时区是否为空
    if (timeZone == nil) {
```



```

        NSAssert(timeZone != nil, @"Can't create timezone: %@",
        closesZoneInfo);
        timeZone = [NSTimeZone systemTimeZone];
    }
    return timeZone;
}

```

importDataBaseFromFile:方法实现对文件进行解析，并获取。程序代码如下：

```

- (NSArray *)importDataBaseFromFile:(NSString *)fileName {
    NSString *filePath = [[NSBundle mainBundle] pathForResource:filename
    ofType:nil]; //创建字符串对象
    NSData *jsonData = [NSData dataWithContentsOfFile:filePath];
    NSAssert(jsonData.length != 0, @"timezonesDB.json not found in app
    bundle");
    NSError *error = nil;
    NSArray *timeZones = [NSJSONSerialization JSONObjectWithData:jsonData
    options:NSJSONReadingAllowFragments error:&error]; //解析
    NSAssert(error == nil, @"JSON parsing failed");
    return timeZones;
}

```

closesZoneInfoWithLocation:方法获取最接近的距离，程序代码如下：

```

- (NSDictionary *)closesZoneInfoWithLocation:(CLLocation *)location source:
(NSArray *)source {
    CLLocationDistance closestDistance = DBL_MAX;
    NSDictionary *closestZoneInfo = nil;
    //遍历
    for (NSDictionary *locationInfo in source) {
        double latitude = [locationInfo[@"latitude"] doubleValue];
        double longitude = [locationInfo[@"longitude"] doubleValue];
        CLLocation *zoneLocation = [[CLLocation alloc] initWithLatitude:
        latitude longitude:longitude];
        CLLocationDistance distance = [location distanceFromLocation:
        zoneLocation];
        //判断是否为最近的距离
        if (distance < closestDistance) {
            closestDistance = distance;
            closestZoneInfo = locationInfo;
        }
    }
    return closestZoneInfo;
}

```

(11) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议以及插座变量的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
//头文件
#import <MapKit/MapKit.h>
#import "TimeZone.h"
@interface ViewController : UIViewController<MKMapViewDelegate>{
    //插座变量
}

```

```
IBOutlet MKMapView *map;
IBOutlet UILabel *timeZoneLabel;
}
@end
```

(12) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 6.20 所示。

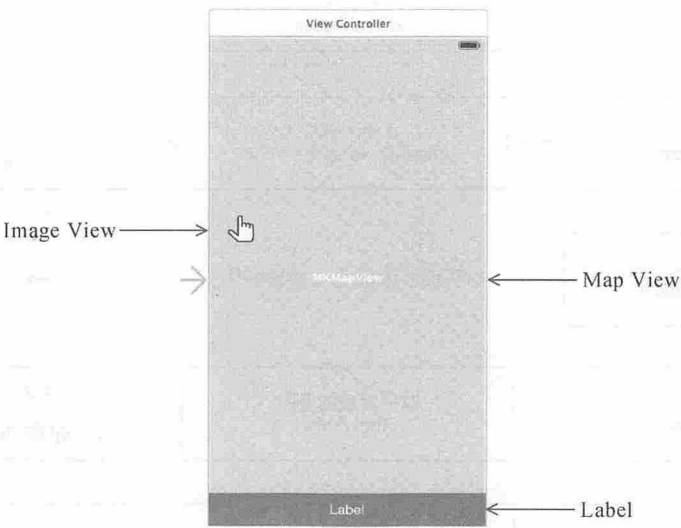


图 6.20 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 6-15 所示。

表 6-15 视图、控件设置

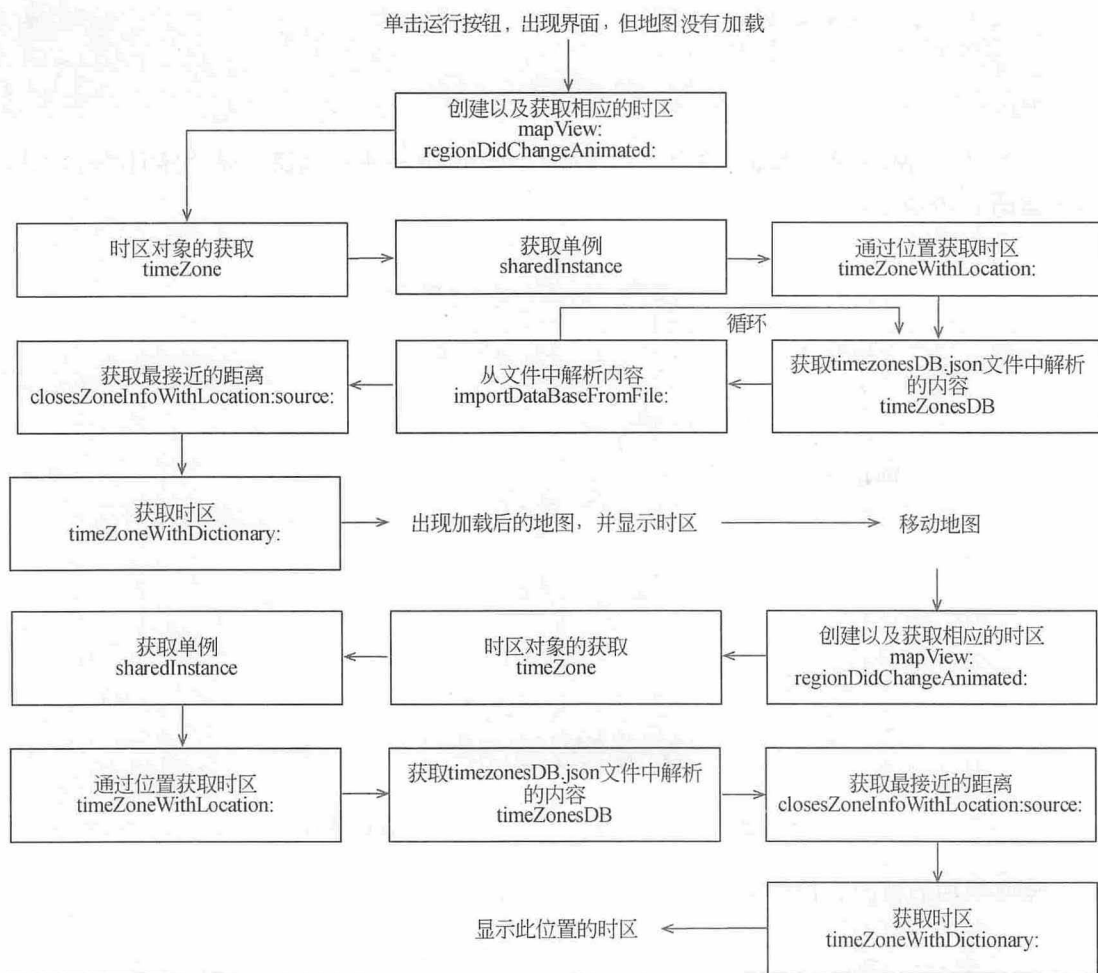
视图、控件	属 性 设 置	其 他
Label1	Font: System 16.0 Alignment: 居中	与插座变量 timeZoneLabel 关联
Map View		与插座变量 map 关联 与 delegate 关联
Image View	Image: 1.png	

(13) 打开 ViewController.m 文件，编写代码实现对 CLLocation 对象的创建以及获取相应的时区。程序代码如下：

```
-(void)mapView:(MKMapView *)mapView regionDidChangeAnimated:(BOOL)
animated {
    CLLocation *lo = [[CLLocation alloc] initWithLatitude:mapView.
    centerCoordinate.latitude longitude:mapView.centerCoordinate.longitude];
    //实例化对象
    timeZoneLabel.text = lo.timeZone.description; //设置文本内容
}
```

【代码解析】

由于本实例的中的代码和方法非常多，为了方便读者的阅读，笔者绘制了它们的执行流程，如图 6.21 所示。



实例 81 自定义地图的标注

【实例描述】

本实例实现的功能是自定义地图上的标注。当用户单击“添加标注”按钮，就会在指定的位置上添加标注，此时的标注是自定义的，其中标注中的大头针变为了一个旗子，单击此旗子后，标注中显示的注释视图也添加了对应的图像以及按钮，并且单击此按钮后，就会进行另一个界面。运行效果如图 6.22 所示。

【实现过程】

- (1) 创建一个项目，命名为“自定义地图的标注”。
- (2) 添加图像 1.png、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 MapKit.framework 到创建的项目中。
- (4) 创建一个基于 NSObject 类的 Annotation 类。



图 6.22 运行效果

(5) 打开 Annotation.h 文件, 编写代码, 实现头文件、遵守协议、对象以及属性的声明。程序代码如下:

```
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>
@interface Annotation : NSObject<MKAnnotation>{
    //对象
    UIImage *image;
    NSNumber *latitude;
    NSNumber *longitude;
}
//属性
@property (nonatomic, retain) UIImage *image;
@property (nonatomic, retain) NSNumber *latitude;
@property (nonatomic, retain) NSNumber *longitude;
@end
```

(6) 打开 Annotation.m 文件, 编写代码, 实现标注的坐标以及注释视图的设置。程序代码如下:

```
@synthesize image;
@synthesize latitude;
@synthesize longitude;
```

```

//获取坐标
- (CLLocationCoordinate2D) coordinate;
{
    CLLocationCoordinate2D theCoordinate;
    theCoordinate.latitude = 37.786996;
    theCoordinate.longitude = -122.419281;
    return theCoordinate;
}
//获取标注的标题
- (NSString *)title
{
    return @"San Francisco";
}
//获取标注的子标题
- (NSString *)subtitle
{
    return @"Founded: June 29, 1776";
}

```

(7) 创建一个基于 UIViewController 类的 CityViewController 类。

(8) 打开 CityViewController.m 文件，编写代码，实现标题的设置。程序代码如下：

```

- (void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    self.title=@"详情";
}

```

(9) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量、对象、方法以及动作的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
//头文件
#import <MapKit/MapKit.h>
#import "Annotation.h"
#import "CityViewController.h"
@interface ViewController : UIViewController{
    IBOutlet MKMapView *map;
    //对象
    NSMutableArray *mapannotation;
    CityViewController *city;
    NSMutableArray *mapAnnotations;
}
//方法
+ (CGFloat)annotationPadding;
+ (CGFloat)calloutHeight;
- (IBAction)biaozhu:(id) sender;
@end

```

(10) 打开 Main.storyboard 文件，添加 Navigation Controller 导航控制器到画布中，将此控制器连接的根视图改为 View Controller 视图控制器。对 View Controller 视图控制器的设计界面进行设计，效果如图 6.23 所示。

需要添加的视图、控件以及对它们的设置如表 6-16 所示。

从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 CityViewController。这时，新增的 View Controller 视图控制器就变为了 City View Controller

视图控制器。在 Identity 面板下, 将 Storyboard ID 设置为 City, 选中 Use Storyboard ID 复选框。从视图库中添加 Image View 图像视图到此控制器的设计界面, 将 Image 设置为 3.jpg, 最后画布的效果如图 6.24 所示。

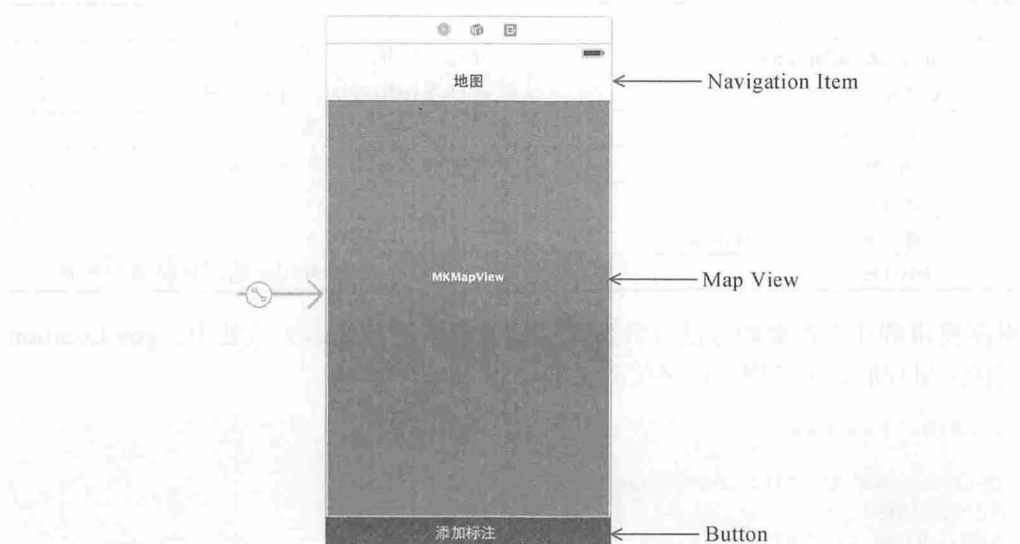


图 6.23 View Controller 视图控制器的设计界面

表 6-16 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 黑色	
Navigation Item	Title: 地图	
Map View		与插座变量 map 关联 与 delegate 关联
Button	Title: 添加标注 Font: System 17.0 Text Color: 黑色	与动作 biaoazu:关联

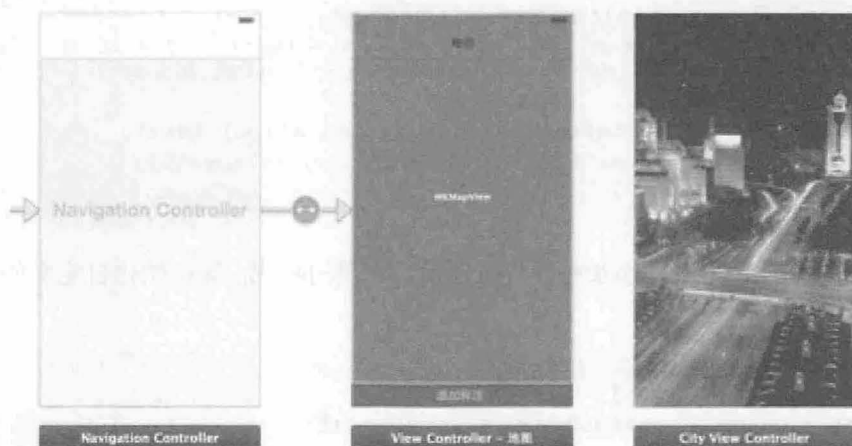


图 6.24 画布效果

(11) 打开 ViewController.m 文件，编写代码，实现在界面显示自定义的标注。使用的方法如表 6-17 所示。

表 6-17 ViewController.m 文件中方法总结

方 法	功 能
annotationPadding	获取标注的填充范围
calloutHeight	获取标注中出现的注释视图的高度
gotoLocation	指定坐标的显示范围
viewDidLoad	视图加载后调用，实现初始化
biaozhu:	单击“添加标注”按钮
mapView:viewForAnnotation:	设置并获取地图的标注
showDetails:	显示 City View Controller 视图控制器的界面

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，gotoLocation 方法实现指定坐标的显示范围。程序代码如下：

```

- (void) gotoLocation
{
    MKCoordinateRegion newRegion;
    //坐标中心
    newRegion.center.latitude = 37.786996;
    newRegion.center.longitude = -122.440100;
    //缩放级别
    newRegion.span.latitudeDelta = 0.112872;
    newRegion.span.longitudeDelta = 0.109863;
    //设置显示位置
    [map setRegion:newRegion animated:YES];
}

```

viewDidLoad 方法在视图加载后调用，实现对栏按钮条目的创建以及初始化，并且创建并设置自定义的标注。程序代码如下：

```

- (void) viewDidLoad
{
    //创建并设置栏按钮条目对象
    UIBarButtonItem *temporaryBarButtonItem = [[UIBarButtonItem alloc]
    init];
    temporaryBarButtonItem.title = @"Back";
    self.navigationItem.backBarButtonItem = temporaryBarButtonItem;
    mapannotation = [[NSMutableArray alloc] initWithCapacity:1];
    //创建并设置
    Annotation *sfAnnotation = [[Annotation alloc] init];
    [mapannotation insertObject:sfAnnotation atIndex:0];
    [self gotoLocation];
}

```

biaozhu:方法实现按钮“添加标注”按钮后，实现的响应，即初始化自定义的标注。程序代码如下：

```

- (IBAction)biaozhu:(id) sender {
    [self gotoLocation];
    [map removeAnnotations:map.annotations];           //移除标注
    [map addAnnotation:[mapannotation objectAtIndex:0]]; //添加标注
}

```

mapView:viewForAnnotation:方法实现对标注的设置以及获取功能。程序代码如下:

```
- (MKAnnotationView *)mapView:(MKMapView *)theMapView viewForAnnotation:
(id <MKAnnotation>)annotation
{
    if ([annotation isKindOfClass:[MKUserLocation class]])
        return nil;
    //处理自定义标注
    if ([annotation isKindOfClass:[Annotation class]]){
        static NSString* SFAnnotationIdentifier = @"SFAnnotationIdentifier";
        MKPinAnnotationView* pinView =(MKPinAnnotationView *) [map
        dequeueReusableAnnotationViewWithIdentifier:SFAnnotationIdentifier];
                                                                    //创建 pinView 对象

        if (!pinView){
            //设置自定义的标注
            MKAnnotationView *annotationView = [[MKAnnotationView alloc]
            initWithAnnotation:annotation reuseIdentifier:SFAnnotationIdentifier];
                                                                    //实例化对象

            annotationView.canShowCallout = YES;
            UIImage *flagImage = [UIImage imageNamed:@"1.png"];
            //尺寸的设置
            CGRect resizeRect;
            resizeRect.size = flagImage.size;                                                                    //获取尺寸
            CGSize maxSize = CGRectInset(self.view.bounds, [ViewController
            annotationPadding], [ViewController annotationPadding]).size;
                                                                    //获取尺寸
            maxSize.height -= self.navigationController.navigationBar.frame.
            size.height + [ViewController calloutHeight];
            //判断 resizeRect 的宽度是否大于最大的宽度
            if (resizeRect.size.width > maxSize.width)
                resizeRect.size = CGSizeMake(maxSize.width, resizeRect.size.
                height / resizeRect.size.width * maxSize.width);
            //判断 resizeRect 的高度是否大于最大的高度
            if (resizeRect.size.height > maxSize.height)
                resizeRect.size = CGSizeMake(resizeRect.size.width /
                resizeRect.size.height * maxSize.height, maxSize.height);
            resizeRect.origin = (CGPoint){0.0f, 0.0f};
            UIGraphicsBeginImageContext(resizeRect.size);
                                                                    //创建基于位图的上下文

            [flagImage drawInRect:resizeRect];
            UIImage *resizedImage = UIGraphicsGetImageFromCurrentImage
            Context();
            UIGraphicsEndImageContext();                                                                    //结束上下文
            annotationView.image = resizedImage;                                                                //设置图像
            annotationView.opaque = NO;
            //为注释视图添加图像
            UIImageView *sfIconView = [[UIImageView alloc] initWithImage:
            [UIImage imageNamed:@"2.png"]];
            annotationView.leftCalloutAccessoryView = sfIconView;
            //为注释视图添加按钮
            UIButton* rightButton = [UIButton buttonWithType:
            UIButtonTypeDetailDisclosure];
            [rightButton addTarget:self action:@selector(showDetails:)
            forControlEvents:UIControlEventTouchUpInside];                                                                    //添加动作
            annotationView.rightCalloutAccessoryView=rightButton;
            return annotationView;
        }
    }
    else
```

```

    {
        pinView.annotation = annotation;           //设置标注
    }
    return pinView;
}
return nil;
}

```

showDetails:方法在单击注释视图中的按钮后，实现切换功能。程序代码如下：

```

- (void) showDetails:(id) sender
{
    city=[self.storyboard instantiateViewControllerWithIdentifier:@"City"];
    //实例化对象
    [self.navigationController pushViewController:city animated:YES];
}

```

【代码解析】

本实例关键功能是标注中大头针的改变、注释视图中图像的添加以及按钮的添加。下面依次讲解这3个知识点。

1. 标注中大头针的改变

MKAnnotationView 的 image 属性可以改变标注中显示大头针的图像，其语法形式如下：

```
@property (nonatomic, retain) UIImage *image;
```

在本实例中，就是使用了 MKAnnotationView 的 image 属性将标注中显示的大头针改变为了旗子。代码如下：

```

UIImage *flagImage = [UIImage imageNamed:@"1.png"];
.....
[flagImage drawInRect:resizeRect];
UIImage *resizedImage = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
annotationView.image = resizedImage;

```

2. 注释视图中图像的添加

MKAnnotationView 的 leftCalloutAccessoryView 属性实现在弹出注释视图后，在左边添加附属视图，其语法形式如下：

```
@property (retain, nonatomic) UIView *leftCalloutAccessoryView;
```

在本实例中，就使用了 MKAnnotationView 的 leftCalloutAccessoryView 属性在注释视图的左边添加了图像视图。程序代码如下：

```

UIImageView *sfIconView = [[UIImageView alloc] initWithImage:[UIImage
imageNamed:@"2.png"]];
annotationView.leftCalloutAccessoryView = sfIconView;

```

3. 注释视图中按钮的添加

MKAnnotationView 的 rightCalloutAccessoryView 属性实现在弹出注释视图后，在右边添加按钮，其语法形式如下：

```
@property (retain, nonatomic) UIView *rightCalloutAccessoryView ;
```

在本实例中,就使用了 MKAnnotationView 的 rightCalloutAccessoryView 属性在注释视图的右边添加了按钮。程序代码如下:

```
UIButton* rightButton = [UIButton buttonWithType:UIButtonTypeDetailDisclosure];
[rightButton addTarget:self action:@selector(showDetails:) forControlEvents:
UIControlEventTouchUpInside];
annotationView.rightCalloutAccessoryView=rightButton;
```

实例 82 自定义的地图

【实例描述】

本实例实现的功能是构建一个自定义的地图,这里没有实现 iOS 规定的地图视图。当单击运行按钮后,显示在界面的是一个自定义的地图,并且在地图上还会出现自定义的标注。运行效果如图 6.25 所示。

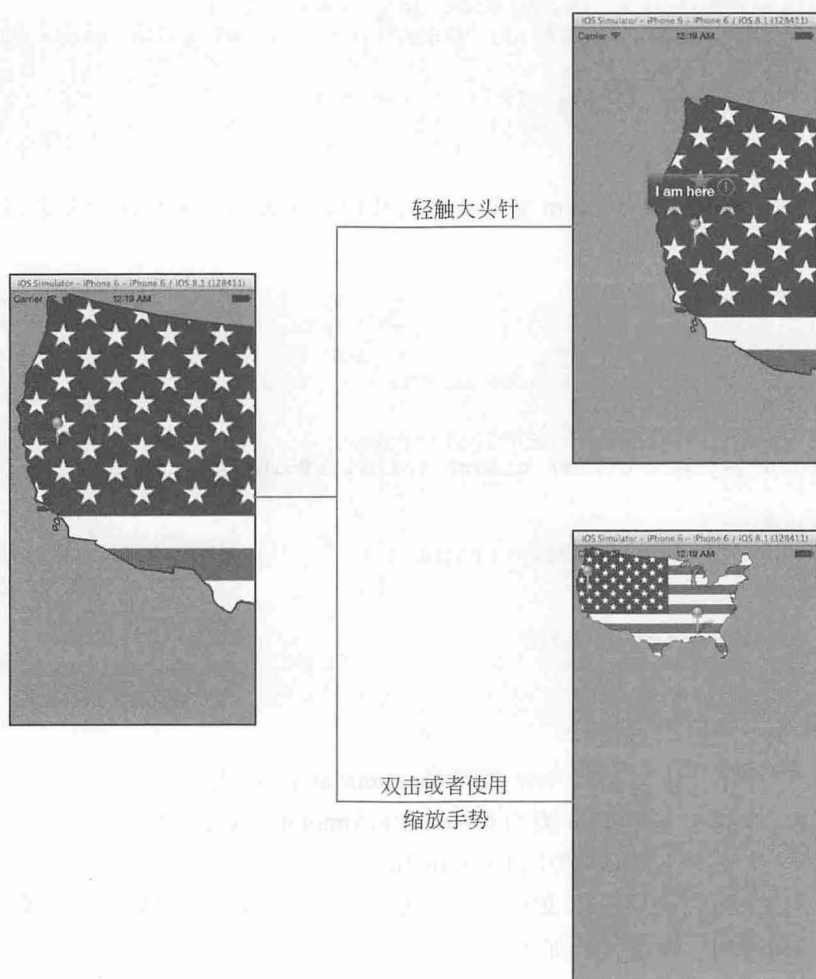


图 6.25 运行效果

【实现过程】

(1) 创建一个项目，命名为“自定义的地图”。

(2) 添加图像 1.png、2.png、3.png、4.png、5.png、6.png 到创建项目的 Supporting Files 文件夹中。

(3) 创建一个基于 NSObject 类的 CustomAnnotation 类。

(4) 打开 CustomAnnotation.h 文件，编写代码，实现实例变量、对象、属性以及方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
@interface CustomAnnotation : NSObject{
    CGPoint _point;
    //对象
    NSString *_title;
    NSString *_subtitle;
    UIButton *_rightCalloutAccessoryView;
}
//属性
@property (nonatomic, assign) CGPoint point;
@property (nonatomic, copy) NSString *title;
@property (nonatomic, copy) NSString *subtitle;
@property (nonatomic, retain) UIButton *rightCalloutAccessoryView;
//方法
+ (id)annotationWithPoint:(CGPoint)point;
- (id)initWithPoint:(CGPoint)point;
@end
```

(5) 打开 CustomAnnotation.m 文件，编写代码，实现自定义标注的获取以及初始化。程序代码如下：

```
@synthesize point = _point;
@synthesize title = _title;
@synthesize subtitle = _subtitle;
@synthesize rightCalloutAccessoryView = _rightCalloutAccessoryView;
//获取使用位置初始化后的标注
+ (id)annotationWithPoint:(CGPoint)point {
    return [[[self class] alloc] initWithPoint:point] ;
}
//初始化标注
- (id)initWithPoint:(CGPoint)point {
    self = [super init];
    if (nil != self) {
        self.point = point;           //设置点
    }
    return self;
}
```

(6) 创建一个基于 UIScrollView 类的 CustomMapView 类。

(7) 创建一个基于 UIButton 类的 CustomPinAnnotationView 类。

(8) 创建一个基于 UIView 类的 CustomView 类。

(9) 打开 CustomMapView.h 文件，编写代码，实现头文件、宏定义、对象、实例变量、属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
//头文件
```

```

@class CustomAnnotation;
#import "CustomPinAnnotationView.h"
#import "CustomAnnotation.h"
@class CustomView;
#define ZOOM_STEP 1.5 //宏定义
@interface CustomMapView : UIScrollView<UIScrollViewDelegate>{
    //对象
    UIImageView *_customMap;
    NSMutableArray *_pinAnnotations;
    CustomView *_callout;
    CGSize _originalSize; //实例变量
}
//属性
@property (nonatomic, retain) UIImageView *customMap;
@property (nonatomic, retain) NSMutableArray *pinAnnotations;
@property (nonatomic, retain) CustomView *callout;
@property (nonatomic, assign) CGSize originalSize;
//方法
- (void)displayMap:(UIImage *)map; //显示地图
- (void)addAnnotation:(CustomAnnotation *)annotation animated:(BOOL)
animate;
- (void)hideCallOut; //隐藏注释视图
- (IBAction)showCallOut:(id)sender;
- (void)centreOnPoint:(CGPoint)point animated:(BOOL)animate;
@end

```

(10) 打开 CustomMapView.m 文件, 编写代码, 实现自定义地图的绘制, 实现地图的放大、缩小, 以及为地图添加标注。使用的方法如表 6-18 所示。

表 6-18 CustomMapView.m 文件中方法总结

方 法	功 能
awakeFromNib	添加手势等功能
handleDoubleTap:	双击功能
handleTwoFingerTap:	缩放手势的功能
displayMap:	显示地图
addAnnotation:animated:	添加标注对象
showCallOut:	显示注释视图
hideCallOut	隐藏注释视图
centreOnPoint:animated:	获取可滚动的区域的偏移量
touchesEnded:withEvent:	触摸结束
viewForZoomingInScrollView:	获取自定义地图

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。自定义地图的绘制需要使用 awakeFromNib、displayMap:方法实现。其中, awakeFromNib 方法实现对轻拍手势识别器对象的添加以及对其他的一些默认设置进行设置, 程序代码如下:

```

- (void)awakeFromNib {
    //设置滚动视图
    self.delegate = self;
    self.showsHorizontalScrollIndicator = NO;
    self.showsVerticalScrollIndicator = NO;
    [self setUserInteractionEnabled:YES];
    //创建并设置轻拍的手势识别器

```

```

UITapGestureRecognizer *doubleTap = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(handleDoubleTap:)];
UITapGestureRecognizer *twoFingerTap = [[UITapGestureRecognizer alloc]
initWithTarget:self action:@selector(handleTwoFingerTap:)];
[doubleTap setNumberOfTapsRequired:2];           //设置轻拍的次数
[twoFingerTap setNumberOfTouchesRequired:2];
//添加
[self addGestureRecognizer:doubleTap];
[self addGestureRecognizer:twoFingerTap];
}

```

displayMap:方法实现对地图的显示。程序代码如下:

```

- (void)displayMap:(UIImage *)map {
    if (!self.customMap) {
        //创建并设置图像视图对象
        UIImageView *imageView = [[UIImageView alloc] initWithImage:map];
        imageView.userInteractionEnabled = YES;
        self.customMap = imageView;
        [self addSubview:self.customMap];           //添加视图对象
    } else {
        self.customMap.image = map;                 //设置图像
    }
    //设置尺寸
    self.originalSize = CGSizeMake(self.customMap.frame.size.width,
self.customMap.frame.size.height);
    self.contentSize = self.originalSize;           //设置滚动视图可滚动的区域
}

```

地图的放大缩小可以以两种方式进行实现，需要使用 handleDoubleTap: 和 handleTwoFingerTap: 方法实现。其中，handleDoubleTap: 方法实现的是第一种缩放方式——双击进行缩放。程序代码如下:

```

- (void)handleDoubleTap:(UIGestureRecognizer *)gestureRecognizer {
    float newScale = self.zoomScale >= self.maximumZoomScale ?
self.minimumZoomScale : self.zoomScale * ZOOM_STEP;
    [self setZoomScale:newScale animated:YES];           //设置缩放以及动画
}

```

handleTwoFingerTap: 方法实现的是第二种缩放方式——缩放手势的功能。程序代码如下:

```

- (void)handleTwoFingerTap:(UIGestureRecognizer *)gestureRecognizer {
    float newScale = self.zoomScale <= self.minimumZoomScale ?
self.maximumZoomScale : self.zoomScale / ZOOM_STEP;
    [self setZoomScale:newScale animated:YES];           //设置缩放以及动画
}

```

为地图添加标注，需要使用 addAnnotation:animated:、showCallOut:、hideCallOut、centreOnPoint:animated:、touchesEnded:animated: 方法。其中，addAnnotation:animated: 方法实现添加标注以及标注的动画效果。

```

- (void)addAnnotation:(CustomAnnotation *)annotation animated:(BOOL)
animate {
    //创建并设置标注的自定义大头针对象
    CustomPinAnnotationView *pinAnnotation = [[CustomPinAnnotationView
alloc] initWithAnnotation:annotation onView:self animated:animate];
    //判断 _pinAnnotations 是否为空
}

```

```

if (!_pinAnnotations) {
    _pinAnnotations = [[NSMutableArray alloc] init];
}
[self.pinAnnotations addObject:pinAnnotation];           //添加对象
[self addObserver:pinAnnotation forKeyPath:@"contentSize" options:
NSKeyValueObservingOptionNew context:nil];             //添加消息通知
}

```

showCallOut:方法实现标注中注释视图的显示功能。程序代码如下:

```

- (IBAction)showCallOut:(id)sender {
    //遍历
    for (CustomPinAnnotationView *pin in self.pinAnnotations) {
        if (pin == sender) {
            //判断
            if (!self.callout) {
                //创建并设置自定义的注释视图
                CustomView * calloutView = [[CustomView alloc]
                initWithAnnotation:pin.annotation onMap:self];
                self.callout = calloutView;
                [self addObserver:self.callout forKeyPath:@"contentSize"
                options:NSKeyValueObservingOptionNew context:nil];
                //添加消失通知

                [self addSubview:self.callout];             //添加视图对象
            } else {
                [self hideCallout];                         //隐藏注释视图
                [self.callout displayAnnotation:pin.annotation];
            }
            [self centreOnPoint:pin.annotation.point animated:YES];
            break;
        }
    }
}

```

(11) 打开 CustomPinAnnotationView.h 文件, 编写代码, 实现头文件、宏定义、对象、属性以及方法的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
//头文件
@class CustomMapView;
#import "CustomAnnotation.h"
//宏定义
#define PIN_WIDTH          32.0
#define PIN_HEIGHT         39.0
#define PIN_POINT_X        8.0
#define PIN_POINT_Y        35.0
#define CALLOUT_OFFSET_X   7.0
#define CALLOUT_OFFSET_Y   5.0
#define Green_PIN           @"2.png"
@interface CustomPinAnnotationView : UIButton{
    CustomAnnotation *_annotation;
}
@property (nonatomic, retain) CustomAnnotation *annotation; //属性
//方法
- (CGRect)frameForPoint:(CGPoint)point;
- (id)initWithAnnotation:(CustomAnnotation *)annotation onView:
(CustomMapView *)mapView animated:(BOOL)animate;
@end

```

(12) 打开 CustomPinAnnotationView.m 文件，编写代码，实现对标注中大头针的自定义。使用的方法如表 6-19 所示。

表 6-19 CustomPinAnnotationView.m文件中方法总结

方 法	功 能
initWithAnnotation: onView: animated:	初始化大头针
frameForPoint:	获取框架
observeValueForKeyPath: ofObject: change: context:	当属性的值发生变化时，自动调用此方法，实现框架的改变

其中，initWithAnnotation: onView: animated:方法实现了对自定义大头针的初始化功能。程序代码如下：

```

- (id)initWithAnnotation: (CustomAnnotation *)annotation onView: (CustomMapView *)mapView animated: (BOOL)animate {
    CGRect frame = CGRectMake(0, 0, 0, 0);
    if ((self = [super initWithFrame:frame])) {
        self.annotation = annotation;           //设置标注
        self.frame = [self frameForPoint:self.annotation.point]; //设置框架

        [self setImage:[UIImage imageNamed:Green_PIN] forState:UIControlStateNormal];
        [self addTarget:mapView action:@selector(showCallOut:) forControlEvents:
        UIControlEventTouchDown];              //添加动作
        if (!self.annotation.title) {
            [self setImage:[UIImage imageNamed:Green_PIN] forState:
            UIControlStateDisabled]; //设置图像
            self.enabled = self.annotation.title ? YES : NO;
        }
        [mapView addSubview:self];              //添加视图对象
        //判断
        if (animate) {
            //下落的动画效果
            CABasicAnimation *pindrop = [CABasicAnimation animationWithKeyPath:
            @"position.y"];
            pindrop.duration = 0.5f;             //设置动作持续时间
            pindrop.fromValue = [NSNumber numberWithFloat:self.center.y - mapView.frame.size.height]; //设置开始值
            pindrop.toValue = [NSNumber numberWithFloat:self.center.y]; //设置结束值
            pindrop.timingFunction = [CAMediaTimingFunction functionWithName:
            kCAMediaTimingFunctionEaseOut];
            [self.layer addAnimation:pindrop forKey:@"pindrop"]; //添加动画
        }
    }
    return self;
}

```

observeValueForKeyPath: ofObject: change: context:方法在属性的值发生变化时，会被自动调用，它实现大头针框架的改变。程序代码如下：

```

- (void)observeValueForKeyPath: (NSString *)keyPath ofObject: (id)object
change: (NSDictionary *)change context: (void *)context {
    //判断 keyPath 是否为"contentSize"
}

```



```

    if ([keyPath isEqual:@"contentSize"]) {
        CustomMapView *mapView = (CustomMapView *)object;
        float width = (mapView.contentSize.width / mapView.originalSize.
width) * self.annotation.point.x;
        float height = (mapView.contentSize.height / mapView.originalSize.
height) * self.annotation.point.y;
        self.frame = [self frameForPoint:CGPointMake(width, height)];
        //设置框架
    }
}

```

(13) 打开 CustomView.h 文件, 编写代码, 实现头文件、宏定义、对象、属性以及方法的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
//头文件
#import "CustomAnnotation.h"
@class CustomMapView;
@class CustomMapView;
//宏定义
#define CALLOUT_HEIGHT 57
#define CALLOUT_LEFT_IMAGE_WIDTH 17
#define CALLOUT_RIGHT_IMAGE_WIDTH 17
#define CALLOUT_MAX_WIDTH 220
.....
#define SUBTITLE_FONT_SIZE 11.0
#define RIGHT_ACCESSORY_LEFT_OFFSET 2.0
#define RIGHT_ACCESSORY_TOP_OFFSET 9.0
@interface CustomView : UIView{
    //对象
    CustomAnnotation *_annotation;
    CustomMapView *_mapView;
}
//属性
@property (nonatomic, retain) CustomAnnotation *annotation;
@property (nonatomic, retain) CustomMapView *mapView;
//方法
- (id)initWithAnnotation:(CustomAnnotation *)annotation onMap:
(CustomMapView *)mapView;
- (void)displayAnnotation:(CustomAnnotation *)annotation;
@end

```

(14) 打开 CustomView.m 文件, 编写代码, 实现对标注中的注释视图进行自定义。使用的方法如表 6-20 所示。

表 6-20 CustomView.m 文件中方法总结

方 法	功 能
calculateCentreWidth	计算中间的宽度
calculateViewWidth	获取视图的宽度
setFramePositionForPoint:	设置框架
setBackgroundImages	设置背景图像
setLabels	设置标签
initWithAnnotation: onMap:	初始化
displayAnnotation:	显示
observeValueForKeyPath:ofObject:change:context:	当属性的值发生变化时, 自动调用此方法, 实现对框架的设置

其中，`calculateCentreWidth` 实现对注释视图中间宽度的计算。程序代码如下：

```
- (float)calculateCentreWidth {
    float width = CALLOUT_ANCHOR_WIDTH;
    NSMutableParagraphStyle *style = [[NSParagraphStyle defaultParagraphStyle]
    mutableCopy];
    [style setLineBreakMode:NSLineBreakByTruncatingTail ]; //设置换行模式
    NSDictionary *dic=[[NSDictionary alloc] initWithObjectsAndKeys:style,
    NSParagraphStyleAttributeName, [UIFont boldSystemFontOfSize:TITLE_
    STANDALONE_FONT_SIZE], NSFontAttributeName, nil];
    //设置标题的尺寸
    CGSize titleSize = [self.annotation.title boundingRectWithSize:CGSizeMake
    (CALLOUT_MAX_WIDTH, TITLE_STANDALONE_LABEL_HEIGHT) options:NSString
    DrawingUsesLineFragmentOrigin | NSStringDrawingUsesFontLeading attributes:
    dic context:nil].size;
    //判断标签尺寸的宽度是否大于 width
    if (titleSize.width > width) {
        width = titleSize.width;
    }
    //判断子标题是否为空
    if (self.annotation.subtitle) {
        //设置副标题的尺寸
        NSMutableParagraphStyle *style = [[NSParagraphStyle
        defaultParagraphStyle] mutableCopy];
        [style setLineBreakMode:NSLineBreakByTruncatingTail ]; //设置换行模式
        NSDictionary *dic=[[NSDictionary alloc] initWithObjectsAndKeys:
        style, NSParagraphStyleAttributeName, [UIFont boldSystemFontOfSize:
        SUBTITLE_FONT_SIZE], NSFontAttributeName, nil];
        //获取尺寸
        CGSize subtitleSize = [self.annotation.subtitle boundingRectWithSize:
        CGSizeMake(CALLOUT_MAX_WIDTH, TITLE_STANDALONE_LABEL_HEIGHT) options:
        NSStringDrawingUsesLineFragmentOrigin | NSStringDrawingUsesFontLeading
        attributes:dic context:nil].size;
        //判断子标题的尺寸是否大于 width
        if (subtitleSize.width > width) {
            width = subtitleSize.width;
        }
    }
    if (self.annotation.rightCalloutAccessoryView) {
        //设置按钮的宽度
        width += self.annotation.rightCalloutAccessoryView.frame.size.
        width + RIGHT_ACCESSORY_LEFT_OFFSET;
    }
    //判断 width 是否大于 CALLOUT_MAX_WIDTH
    if (width > CALLOUT_MAX_WIDTH) {
        return CALLOUT_MAX_WIDTH;
    }
    return width;
}
```

`setFramePositionForPoint`: 方法实现对注释视图框架的设置。程序代码如下：

```
- (void)setFramePositionForPoint:(CGPoint)point {
    float positionX = (self.mapView.contentSize.width / self.mapView.
    originalSize.width) * self.annotation.point.x;
```

```

float positionY = (self.mapView.contentSize.height / self.mapView.
originalSize.height) * self.annotation.point.y;
float calloutWidth = [self calculateViewWidth]; //获取视图的宽度
float x          = positionX - (calloutWidth / 2) - 1;
float y          = positionY - CALLOUT_ANCHOR_HEIGHT - 25.0;
self.frame = CGRectMake(round(x), round(y), calloutWidth, CALLOUT_
ANCHOR_HEIGHT); //设置框架
return;
}

```

setBackgroundImages 方法设置注释视图的背景。程序代码如下:

```

- (void)setBackgroundImages {
    if (!CALLOUT_LEFT_CAP) {
        //设置注释视图最左边的图像
        CALLOUT_LEFT_CAP = [UIImage imageNamed:CALLOUT_LEFT_IMAGE];
    }
    if (!CALLOUT_RIGHT_CAP) {
        //设置注释视图最右边的图像
        CALLOUT_RIGHT_CAP = [UIImage imageNamed:CALLOUT_RIGHT_IMAGE];
    }
    if (!CALLOUT_ANCHOR) {
        //设置注释视图锚点的图像
        CALLOUT_ANCHOR = [UIImage imageNamed:CALLOUT_ANCHOR_IMAGE];
    }
    if (!CALLOUT_BG) {
        //设置注释视图的背景图像
        CALLOUT_BG = [[UIImage imageNamed:CALLOUT_BG_IMAGE]
stretchableImageWithLeftCapWidth:0 topCapHeight:0];
    }
    float centreWidth      = [self calculateCentreWidth];
    float centreOffsetWidth = (centreWidth - CALLOUT_ANCHOR_WIDTH) / 2;
    //创建并设置图像视图, 用来放置注释视图最左边的图像
    UIImageView *leftCap = [[UIImageView alloc] initWithFrame:CGRectMake(0,
0, CALLOUT_LEFT_IMAGE_WIDTH, CALLOUT_HEIGHT)];
    leftCap.image = CALLOUT_LEFT_CAP; //设置图像
    leftCap.alpha = CALLOUT_ALPHA; //设置透明度
    [self addSubview:leftCap];
    //创建并设置图像视图, 用来放置注释视图最右边的图像
    UIImageView *rightCap = [[UIImageView alloc] initWithFrame:CGRectMake
(CALLOUT_LEFT_IMAGE_WIDTH + centreWidth, 0, CALLOUT_RIGHT_IMAGE_WIDTH,
CALLOUT_HEIGHT)];
    rightCap.image = CALLOUT_RIGHT_CAP; //设置图像
    rightCap.alpha = CALLOUT_ALPHA; //设置透明度
    [self addSubview:rightCap];
    //创建并设置图像视图, 用来放置注释视图锚点的图像
    UIImageView *anchor = [[UIImageView alloc] initWithFrame:CGRectMake
(CALLOUT_LEFT_IMAGE_WIDTH + centreOffsetWidth, 0, CALLOUT_ANCHOR_WIDTH,
CALLOUT_ANCHOR_HEIGHT)];
    anchor.image = CALLOUT_ANCHOR; //设置图像
    anchor.alpha = CALLOUT_ALPHA; //设置透明度
    [self addSubview:anchor];
    //判断
    if (centreWidth > CALLOUT_ANCHOR_WIDTH) {
        //设置左右两边的背景
        CGRect leftFrame = CGRectMake(CALLOUT_LEFT_IMAGE_WIDTH, 0,
centreOffsetWidth, CALLOUT_HEIGHT);
        CGRect rightFrame = CGRectMake(CALLOUT_LEFT_IMAGE_WIDTH +

```

```

        centreWidth - centreOffsetWidth, 0, centreOffsetWidth, CALLOUT_HEIGHT);
    UIImageView *leftBG = [[UIImageView alloc] initWithFrame:
        leftFrame];
    leftBG.image = CALLOUT_BG; //设置图像
    leftBG.alpha = CALLOUT_ALPHA; //设置透明度
    [self addSubview:leftBG];
    UIImageView *rightBG = [[UIImageView alloc] initWithFrame:
        rightFrame];
    rightBG.image = CALLOUT_BG; //设置图像
    rightBG.alpha = CALLOUT_ALPHA; //设置透明度
    [self addSubview:rightBG];
    }
}

```

setLabels 方法实现对标签的设置。程序代码如下：

```

- (void)setLabels {
    float titleTopOffset = TITLE_STANDALONE_TOP_OFFSET;
    float titleLabelHeight = TITLE_STANDALONE_LABEL_HEIGHT;
    float titleFontSize = TITLE_STANDALONE_FONT_SIZE;
    float labelWidth = [self calculateCentreWidth]; //获取视图的宽度
    //判断在标注的右边是否有视图
    if (self.annotation.rightCalloutAccessoryView) {
        float x = labelWidth - self.annotation.rightCalloutAccessoryView.
            frame.size.width + CALLOUT_LEFT_IMAGE_WIDTH + RIGHT_ACCESSORY_
            LEFT_OFFSET;
        labelWidth -= self.annotation.rightCalloutAccessoryView.frame.
            size.width;
        self.annotation.rightCalloutAccessoryView.frame = CGRectMake(x,
            RIGHT_ACCESSORY_TOP_OFFSET, self.annotation.rightCalloutAccessoryView.
            frame.size.width, self.annotation.rightCalloutAccessoryView.frame.size.
            height); //设置框架
        [self addSubview:self.annotation.rightCalloutAccessoryView];
    }
    //判断在标注中是否有子标题
    if (self.annotation.subtitle) {
        titleTopOffset = TITLE_TOP_OFFSET;
        titleLabelHeight = TITLE_LABEL_HEIGHT;
        titleFontSize = TITLE_FONT_SIZE;
        //创建并设置标签对象，用来显示子标题的内容
        UILabel *subtitleLabel = [[UILabel alloc] initWithFrame:CGRectMakeMake
            (CALLOUT_LEFT_IMAGE_WIDTH, SUBTITLE_TOP_OFFSET, labelWidth, SUBTITLE_
            LABEL_HEIGHT)];
        subtitleLabel.textColor = [UIColor whiteColor]; //设置文本颜色
        subtitleLabel.backgroundColor = [UIColor clearColor];
        subtitleLabel.font = [UIFont systemFontOfSize:SUBTITLE_FONT_SIZE];
        subtitleLabel.text = self.annotation.subtitle;
        //设置文本内容
        [self addSubview:subtitleLabel];
    }
    //创建并设置标签对象，用来显示标题的内容
    UILabel *titleLabel = [[UILabel alloc] initWithFrame:CGRectMakeMake(CALLOUT_
        LEFT_IMAGE_WIDTH, titleTopOffset, labelWidth, titleLabelHeight)];
    titleLabel.textColor = [UIColor whiteColor];
}

```

```

titleLabel.backgroundColor = [UIColor clearColor];    //设置背景颜色
titleLabel.font = [UIFont boldSystemFontOfSize:titleFontSize];
titleLabel.text = self.annotation.title;    //设置文本内容
[self addSubview:titleLabel];
}

```

displayAnnotation:方法实现注释视图的显示。程序代码如下:

```

- (void)displayAnnotation:(CustomAnnotation *)annotation {
    //遍历
    for (UIView *view in self.subviews) {
        [view removeFromSuperview];    //移除指定的视图对象
    }
    self.annotation = annotation;    //设置标注
    [self setFramePositionForPoint:annotation.point];
    [self setBackgroundImages];
    [self setLabels];    //设置标签
    self.hidden = NO;
    //缩放的动画效果
    CABasicAnimation *scale = [CABasicAnimation animationWithKeyPath:
@"transform.scale"];
    scale.duration = 0.1f;    //设置动作持续时间
    scale.autoreverses = NO;
    scale.removedOnCompletion = YES;
    scale.fromValue = [NSNumber numberWithFloat:0.0f];    //设置开始值
    scale.toValue = [NSNumber numberWithFloat:1.0f];    //设置结束值
    scale.fillMode = kCAFillModeForwards;
    [self.layer addAnimation:scale forKey:@"scale"];
}

```

(15) 打开 ViewController.h 文件, 编写代码, 实现头文件、插座变量的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
@class CustomMapView;
@interface ViewController : UIViewController{
    IBOutlet CustomMapView *mapView;
}
@end

```

(16) 打开 Main.storyboard 文件, 在视图库中拖动 Scroll View 滚动视图到设计界面中, 将此视图的 Class 设置为 CustomMapView。将 Zoom 中的 Min 设置为 0.3, Max 设置为 1.5。将此视图和插座变量 mapView 关联。

(17) 打开 ViewController.m 文件, 编写代码, 实现地图的显示。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [mapView displayMap:[UIImage imageNamed:@"1.png"]];
    mapView.backgroundColor = [UIColor colorWithRed:0.0f green:0.475f

```



```

blue:0.761 alpha:1.0f];
//创建并设置自定义的标注对象（此标注有标题以及子标题）
CustomAnnotation * melbourne = [CustomAnnotation annotationWithPoint:
CGPointMake(543, 380)];
melbourne.title = @"I am here"; //设置标题
melbourne.subtitle = @"2014 年 5 月 5 号"; //设置子标题
[mapView addAnnotation:melbourne animated:NO];
//创建并设置自定义的标注对象（此标注只有标题）
CustomAnnotation * perth = [CustomAnnotation annotationWithPoint:
CGPointMake(63, 200)];
perth.title = @"I am here"; //设置标题
perth.rightCalloutAccessoryView = [UIButton buttonWithType:
UIButtonTypeDetailDisclosure];
[mapView addAnnotation:perth animated:YES];
}

```

【代码解析】

由于本实例中的代码和方法非常多，为了方便读者的阅读，笔者绘制了一些执行流程图如图 6.26～图 6.28 所示。其中，单击运行按钮到自定义地图的显示，使用了 ViewDidLoad、awakeFromNib、displayMap:、addAnnotation:animated:、initWithPoint:、annotationWithPoint:、initWithAnnotation: onView:animated:、frameForPoint:方法共同实现。它们的执行流程如图 6.26 所示。

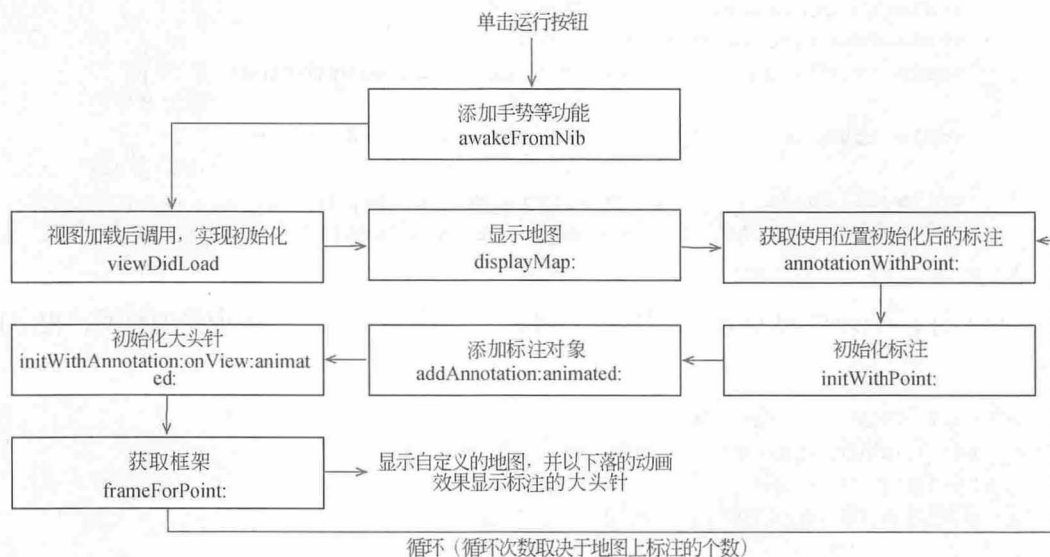


图 6.26 程序执行流程

轻拍地图上的大头针，就会出现注释视图，当轻拍别处，显示的注释视图就会消失，需要使用 showCallOut:、hideCallOut、centreOnPoint:animated:、touchesEnded:withEvent:、viewForZoomingInScrollView:、calculateCentreWidth、calculateViewWidth、setBackgroundImages、setLabels、initWithAnnotation: onMap:、displayAnnotation:方法共同实现。它们的执行流程如图 6.27 所示。

地图的放大缩小有两种方法，以第一种方法为例，双击可实现放大缩小地图，它的实现需要使用 handleDoubleTap:、hideCallOut、touchesEnded:withEvent:、viewForZoomingInScrollView:、observeValueForKeyPath:ofObject:change:context:（CustomPinAnnotationView.m）、frameForPoint:方法共同实现。它们的执行流程如图 6.28 所示。



图 6.27 程序执行流程

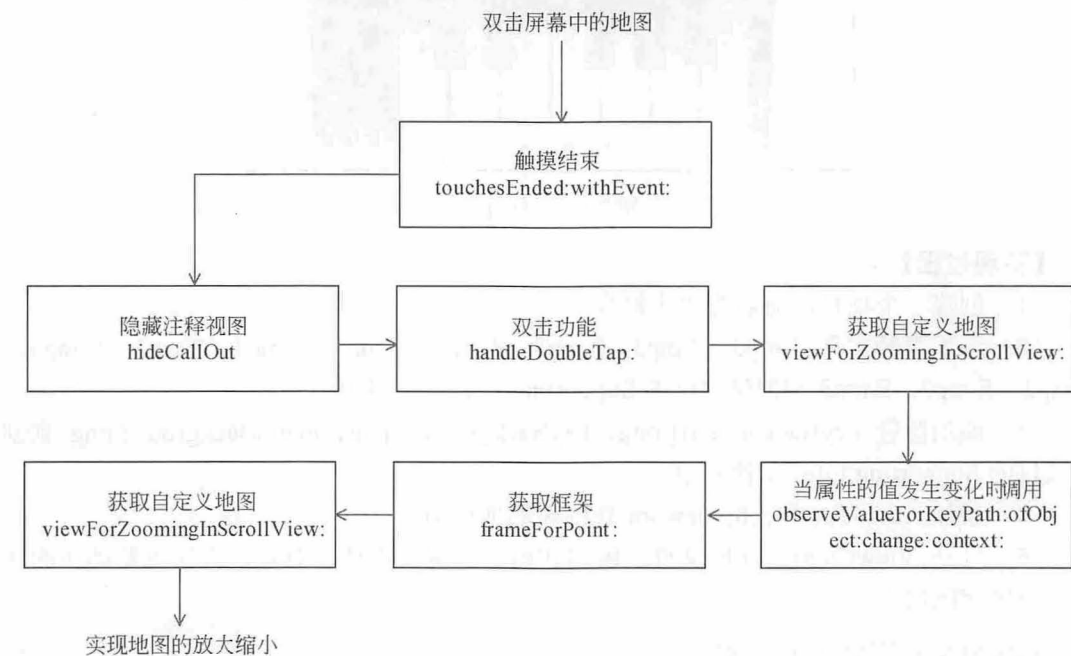


图 6.28 程序执行流程

第7章 音频和视频

音频和视频可以提高用户在听觉以及视觉上的享受。现在很多的软件就是专门为音频和视频开发的，如 QQ 音乐播放器、视频播放器等。本章将主要讲解关于音频和视频的相关实例。

实例 83 小 钢 琴

【实例描述】

艺术源于生活，本实例实现的就是一个具有艺术气息的实例。它是一个小钢琴，当用户按下钢琴中的某一个键时，就会发出相应的响声。运行效果如图 7.1 所示。

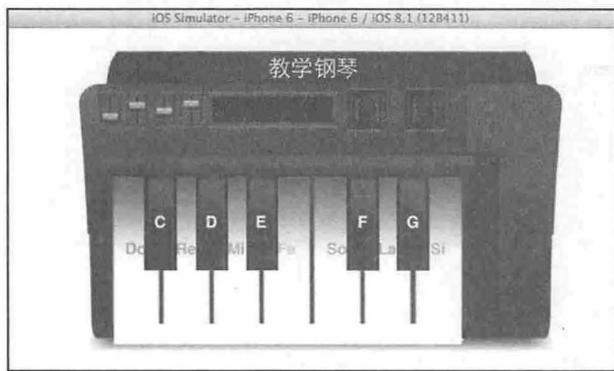


图 7.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“小钢琴”。
- (2) 添加音频文件 1.mp3、2.mp3、3.mp3、4.mp3、5.mp3、6.mp3、7.mp3、C.mp3、D.mp3、E.mp3、F.mp3 到创建项目的 Supporting Files 文件夹中。
- (3) 添加图像 keyBackground1.png、keyBackground2.png、pianoBackground.png 到创建项目的 Supporting Files 文件夹中。
- (4) 添加 AudioToolbox.framework 到创建的项目中。
- (5) 打开 ViewController.h 文件，编写代码，实现头文件、对象、方法以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AudioToolbox/AudioToolbox.h> //头文件
@interface ViewController : UIViewController{
```

```

NSString *soundFile;                                //NSString 对象
}
- (void)playSound:(NSString*) soundKey;              //播放音频文件的方法
//动作
- (IBAction)Do:(id) sender;                          //播放 1.mp3 的声音
- (IBAction)Re:(id) sender;
- (IBAction)Mi:(id) sender;
- (IBAction)Fa:(id) sender;                          //播放 4.mp3 的声音
- (IBAction)So:(id) sender;
- (IBAction)La:(id) sender;
- (IBAction)Si:(id) sender;                          //播放 7.mp3 的声音
- (IBAction)C:(id) sender;
- (IBAction)D:(id) sender;
- (IBAction)E:(id) sender;
- (IBAction)F:(id) sender;
- (IBAction)G:(id) sender;                          //播放 G.mp3 的声音
@end

```

(6) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 7.2 所示。

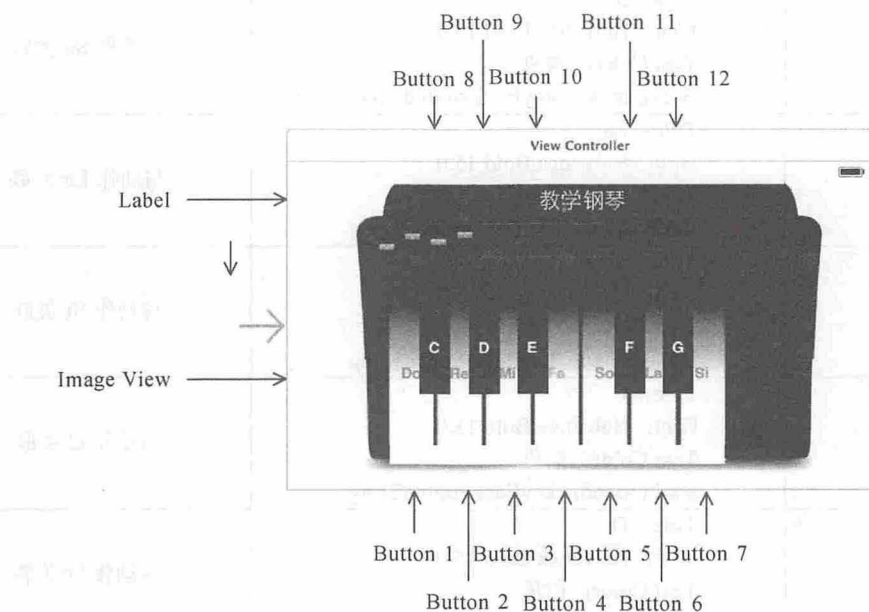


图 7.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-1 所示。

表 7-1 视图、控件设置

视图、控件	属性设置	其他
View Controller	Orientation: Landscape	
Label	Text: 教学钢琴 Color: 白色 Font: System 21.0 Alignment: 居中	
Image View	Image: pianoBackground.png	

续表

视图、控件	属 性 设 置	其 他
Button1	Title: Do Font: Helvetica Bold 15.0 Text Color: 灰色 Background: keyBackground1.png	与动作 Do:关联
Button2	Title: Re Font: Helvetica Bold 15.0 Text Color: 灰色 Background: keyBackground1.png	与动作 Re:关联
Button3	Title: Mi Font: Helvetica Bold 15.0 Text Color: 灰色 Background: keyBackground1.png	与动作 Mi:关联
Button4	Title: Fa Font: Helvetica Bold 15.0 Text Color: 灰色 Background: keyBackground1.png	与动作 Fa:关联
Button5	Title: So Font: Helvetica Bold 15.0 Text Color: 灰色 Background: keyBackground1.png	与动作 So:关联
Button6	Title: La Font: Helvetica Bold 15.0 Text Color: 灰色 Background: keyBackground1.png	与动作 La:关联
Button7	Title: Si Font: Helvetica Bold 15.0 Text Color: 灰色 Background: keyBackground1.png	与动作 Si:关联
Button8	Title: C Font: Helvetica Bold 15.0 Text Color: 白色 Background: keyBackground2.png	与动作 C:关联
Button9	Title: D Font: Helvetica Bold 15.0 Text Color: 白色 Background: keyBackground2.png	与动作 D:关联
Button10	Title: E Font: Helvetica Bold 15.0 Text Color: 白色 Background: keyBackground2.png	与动作 E:关联
Button11	Title: F Font: Helvetica Bold 15.0 Text Color: 白色 Background: keyBackground2.png	与动作 F:关联
Button12	Title: G Font: Helvetica Bold 15.0 Text Color: 白色 Background: keyBackground2.png	与动作 G:关联

(7) 打开 ViewController.m 文件, 编写代码, 实现单击按钮播放相应声音, 从而实现钢琴的声音。使用的方法如表 7-2 所示。

表 7-2 ViewController.m文件中方法总结

方 法	功 能
playSound:	创建系统声音, 并播放
Do:	播放 1.mp3 的声音
Re:	播放 2.mp3 的声音
Mi:	播放 3.mp3 的声音
Fa:	播放 4.mp3 的声音
So:	播放 5.mp3 的声音
La:	播放 6.mp3 的声音
Si:	播放 7.mp3 的声音
C:	播放 C.mp3 的声音
D:	播放 D.mp3 的声音
E:	播放 E.mp3 的声音
F:	播放 F.mp3 的声音
G:	播放 G.mp3 的声音

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, playSound:方法实现的是对系统声音进行创建并播放。程序代码如下:

```
-(void)playSound:(NSString*)soundKey{
    NSString *path = [NSString stringWithFormat:@"%s", [[NSBundle
mainBundle] resourcePath],soundKey];           //创建字符串对象
    SystemSoundID soundID;
    NSURL *filePath = [NSURL fileURLWithPath:path isDirectory:NO];
    AudioServicesCreateSystemSoundID((__bridge CFURLRef)filePath, &soundID);
                                                    //创建一个系统声音
    AudioServicesPlaySystemSound(soundID);        //播放系统声音
}
```

Do:方法实现单击 Do 按钮, 播放 1.mp3 的声音。程序代码如下:

```
-(IBAction)Do:(id)sender {
    soundFile = [NSString stringWithFormat:@"%s/1.mp3"];
    [self playSound: soundFile];                //播放声音
}
```

【代码解析】

本实例关键功能是较短音频的播放。下面就是这个知识点的详细讲解。

在 iOS 一般较短的音频文件播放可以使用 AudioToolbox.framework 框架中的系统声音服务来实现, 它们往往属于系统声音。其中, 要使用系统声音服务, 首先需要使用 AudioServicesCreateSystemSoundID 函数实现系统声音服务对象的创建。其语法形式如下:

```
OSStatus AudioServicesCreateSystemSoundID (
    CFURLRef    inFileURL,
    SystemSoundID *outSystemSoundID
);
```

其中, CFURLRef inFileURL 表示播放的音频文件的 URL; outSystemSoundID 是一个系统 ID, 它在输出时, 表示一个系统声音对象与指定音频文件的关联。在本实例就使用了

AudioServicesCreateSystemSoundID 函数实现系统声音服务对象的创建。代码如下：

```
AudioServicesCreateSystemSoundID((__bridge CFURLRef)filePath, &soundID);
```

其中, (__bridge CFURLRef)filePath 表示播放的音频文件的 URL; &soundID 是一个系统 ID。创建好系统声音服务对象以后就可以对音频文件进行播放了, 需要使用 AudioServicesPlaySystemSound 函数实现, 其语法形式如下:

```
void AudioServicesPlaySystemSound (
    SystemSoundID inSystemSoundID
);
```

其中, SystemSoundID inSystemSoundID 表示系统 ID。在本实例中就使用了 AudioServicesPlaySystemSound 函数实现音频的播放功能, 代码如下:

```
AudioServicesPlaySystemSound(soundID);
```

其中, soundID 表示系统 ID。

实例 84 手机铃声变化器

【实例描述】

本实例实现的是一个铃声变化器的功能, 当用户选择某一铃声后, 相应的铃声就会响起。运行效果如图 7.3 所示。

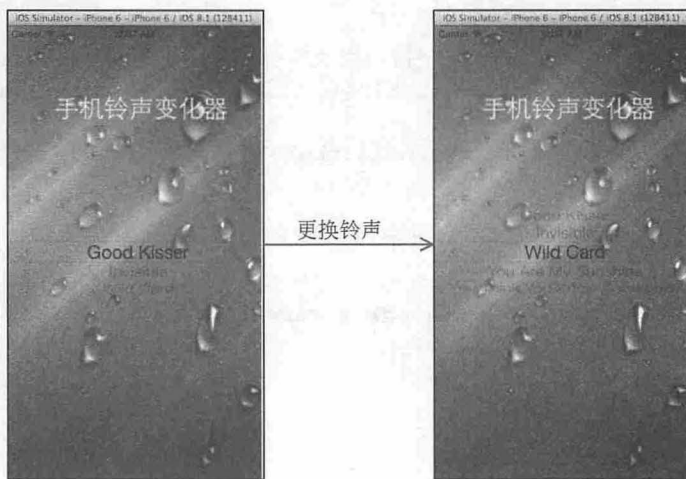


图 7.3 运行效果

【实现过程】

- (1) 创建一个项目, 命名为“手机铃声变化器”。
- (2) 添加音频文件 Good Kiss.mp3、Invisible.mp3、Wild Card.mp3、You Are My Sunshine.mp3、You Think You Know Somebody.mp3 到创建项目的 Supporting Files 文件夹中。
- (3) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。

(4) 添加 AVFoundation.framework 到创建的项目中。

(5) 打开 ViewController.h 文件, 编写代码, 实现头文件、对象的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface ViewController : UIViewController{
    //声明对象
    AVAudioPlayer *audioplayer1;
    AVAudioPlayer *audioplayer2;
    AVAudioPlayer *audioplayer3;
    AVAudioPlayer *audioplayer4;
    AVAudioPlayer *audioplayer5;
    NSArray *array;
}
@end
```

(6) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 7.4 所示。

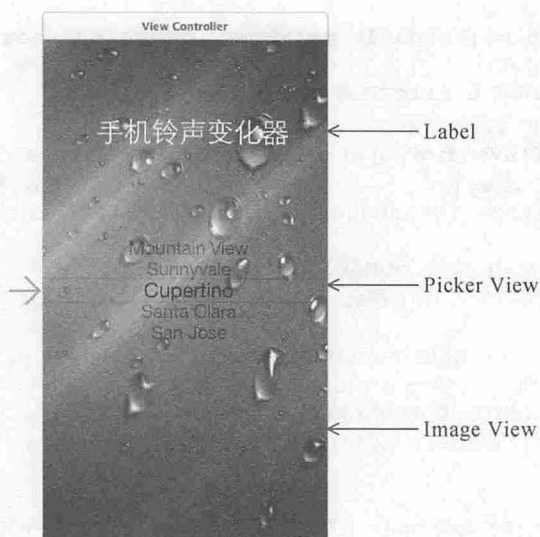


图 7.4 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-3 所示。

表 7-3 视图、控件设置

视图、控件	属性设置	其他
Image View	Image: 1.png	
Label	Text: 手机铃声变化器 Color: 白色 Font: System 31.0 Alignment: 居中	
Picker View		dataSource 与 View Controller 关联 delegate 与 View Controller 关联

(7) 打开 ViewController.m 文件, 编写代码, 实现选择某一铃声就播放此铃声的功能。

使用的方法如表 7-4 所示。

表 7-4 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
numberOfComponentsInPickerView:	获取块数
pickerView:numberOfRowsInComponent:	获取行数
pickerView:titleForRow:forComponent:	获取内容
pickerView:didSelectRow: inComponent:	实现选择

其中，viewDidLoad、numberOfComponentsInPickerView:、pickerView:numberOfRowsInComponent:、pickerView:titleForRow:forComponent:方法实现对自定义选择器的内容填充以及初始化。程序代码如下：

```
- (void)viewDidLoad
{
    array=[NSArray alloc] initWithObjects:@"Good Kisser",@"Invisible",
    @"Wild Card",@"You Are My Sunshine",@"You Think You Know Somebody",nil];
    //创建数组

    [super viewDidLoad];
    NSString *path1=[[NSBundle mainBundle]pathForResource:@"Good Kisser"
    ofType:@"mp3"];
    NSURL *url1=[NSURL fileURLWithPath:path1];
    //创建 AVAudioPlayer 对象
    audioplayer1=[[AVAudioPlayer alloc] initWithContentsOfURL:url1 error:nil];
    [audioplayer1 play]; //播放声音 Good Kisser.mp3
    NSString *path2=[[NSBundle mainBundle]pathForResource:@"Invisible"
    ofType:@"mp3"];
    NSURL *url2=[NSURL fileURLWithPath:path2];
    audioplayer2=[[AVAudioPlayer alloc] initWithContentsOfURL:url2 error:nil];
    .....
    NSString *path5=[[NSBundle mainBundle]pathForResource:@"You Think You
    Know Somebody" ofType:@"mp3"];
    NSURL *url5=[NSURL fileURLWithPath:path5];
    audioplayer5=[[AVAudioPlayer alloc] initWithContentsOfURL:url5 error:nil];
}
//获取块数
-(NSInteger)numberOfComponentsInPickerView: (UIPickerView *)pickerView
{
    return 1;
}
//获取行数
-(NSInteger)pickerView: (UIPickerView *)pickerView numberOfRowsInComponent:
(NSInteger) component
{
    return [array count];
}
//填充内容
- (NSString *)pickerView: (UIPickerView *)pickerView titleForRow: (NSInteger)
row forComponent:
(NSInteger) component {
    return [array objectAtIndex:row];
}
```

pickerView:didSelectRow:方法实现自定义选择器的选择，即选择某一行后，播放此行所显示内容的音频。程序代码如下：


```

-(void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
    inComponent:(NSInteger)component{
    int i=[pickerView selectedRowInComponent:0];
    //判断并进行相应音频的播放
    //判断 i 是否等于 0
    if(i==0){
        //暂停播放的声音
        [audioplayer2 pause];
        [audioplayer3 pause];
        [audioplayer4 pause];
        [audioplayer5 pause];
        if (![audioplayer1 isPlaying])
        {
            [audioplayer1 play];           //播放 audioplayer1 中的声音
        }
    }
    .....
    //判断 i 是否等于 4
    if(i==4){
        //暂停播放的声音
        [audioplayer1 pause];
        [audioplayer2 pause];
        [audioplayer3 pause];
        [audioplayer4 pause];
        if (![audioplayer5 isPlaying])
        {
            [audioplayer5 play];           //播放 audioplayer2 中的声音
        }
    }
}

```

【代码解析】

本实例关键功能是较长音频的播放以及暂停。下面依次讲解这两个知识点。

1. 较长音频的播放

在 iOS 中较长的音频一般会使用 AVAudioPlayer 类来处理。在使用它时，同样是需要进行创建并初始化，它的对象创建初始化方法有很多，一般使用 AVAudioPlayer 的 initWithContentsOfURL:error:实现。其语法形式如下：

```

- (id)initWithContentsOfURL:(NSURL *)url error:(NSError **)outError;

```

其中，(NSURL *)url 表示 NSURL 对象；(NSError **)outError 表示如果发生错误是否返回此错误。在本实例中就使用了 AVAudioPlayer 类来处理较长的音频，它对象的创建初始化使用了 AVAudioPlayer 的 initWithContentsOfURL:error:，代码如下：

```

audioplayer1=[[AVAudioPlayer alloc] initWithContentsOfURL:url1 error:nil];

```

其中，url1 表示 NSURL 对象；nil 表示将错误忽略。创建好 AVAudioPlayer 对象后，就可以对音频进行播放了，需要使用 AVAudioPlayer 的 play 方法实现。

```

- (BOOL)play ;

```

其中，该方法的返回类型为布尔类型。其中，YES 表示播放成功；NO 表示播放失败。在本实例中就使用了 play 方法实现了音频文件的播放，代码如下：


```
[audioplayer1 play];
```

2. 较长音频的暂停

较长音频的暂停可以使用 AVAudioPlayer 的 pause 方法实现，其语法形式如下：

```
- (void)pause;
```

在本实例中就使用了 AVAudioPlayer 的 pause 方法实现了音频文件的播放，代码如下：

```
[audioplayer2 pause];
```

实例 85 十种语言

【实例描述】

在 iOS 7 中新增了 AVSpeechSynthesizer 来实现 Siri 上集成的语言混合功能。本实例就使用此类，实现了使用十种语言说出“我爱你”这句话。当用户单击 Speak 按钮后，就会开始一次说出“我爱你”。运行效果如图 7.5 所示。



图 7.5 运行效果

【实现过程】

- (1) 创建一个项目，命名为“十种语言”。
- (2) 添加 AVFoundation.framework 到创建的项目中。
- (3) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量以及动作的声明。

程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface ViewController : UIViewController{
    IBOutlet UILabel *label; //声明插座变量
}
- (IBAction)speak:(id)sender;
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，

效果如图 7.6 所示。

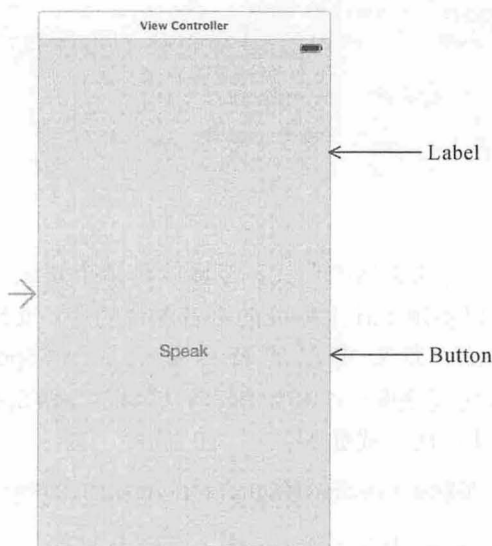


图 7.6 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-5 所示。

表 7-5 视图、控件设置

视图、控件	属性设置	其他
设计界面	Background: 浅灰色	
Label	Text: (空) Font: System 23.0 Alignment: 居中	
Button	Title: Speak Font: System 19.0	与动作 speak:关于

(5) 打开 ViewController.m 文件, 编写代码, 实现使用十种语言说话。程序代码如下:

```

- (IBAction)speak:(id)sender {
    NSString *plistPath = [[NSBundle mainBundle] pathForResource:@"
    Language" ofType:@"plist"];
    NSDictionary *dictionary = [[NSDictionary alloc] initWithContents-
    Offile:plistPath]; //创建字典对象
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_
    DEFAULT, 0), ^{
        for (int i=0; i<[dictionary count]; i++) {
            NSDictionary *dic = [dictionary objectForKey:[NSString
            stringWithFormat:@"%d",i]];
            //创建说话的语言
            AVSpeechSynthesisVoice *voice = [AVSpeechSynthesisVoice
            voiceWithLanguage:[dic objectForKey:@"local"]];
            AVSpeechSynthesizer *av = [[AVSpeechSynthesizer alloc]init];
            //创建语音合成器
            AVSpeechUtterance *utterance = [[AVSpeechUtterance alloc]initWithString:
            [dic objectForKey:@"content"]]; //创建实例化发声的对象
            utterance.voice = voice; //指定语音
        }
    });
}

```

```

utterance.rate *= 0.7; //朗诵速度
[av speakUtterance:utterance];
dispatch_async(dispatch_get_main_queue(), ^{
    label.text = [dic objectForKey:@"name"]; //设置文本中的内容
});
[NSThread sleepForTimeInterval:3];
}
});
}

```

【代码解析】

本实例关键功能是十种语言的说话的形式。下面就是这个知识点的详细讲解。

AVSpeechSynthesizer 可以实现 Siri 上集成的语言混合功能。要想使用此类，首先需要使用 AVSpeechSynthesisVoice 类来实现的特殊语音，AVSpeechSynthesisVoice 的 voiceWithLanguage:方法可以实现获取一个指定语言和区域的 AVSpeechSynthesisVoice 对象，从而实现特定的语音。其语法形式如下：

```
+ (AVSpeechSynthesisVoice *)voiceWithLanguage:(NSString *)language;
```

其中，(NSString *)language 表示为语音指定语言和区域。在本实例中就使用了 voiceWithLanguage:方法实现了十种语音对象的获取。代码如下：

```
AVSpeechSynthesisVoice *voice = [AVSpeechSynthesisVoice voiceWithLanguage:
[dict objectForKey:@"local"]];
```

其中，dict objectForKey:@"local"表示为语音指定语言和区域。接着需要使用 AVSpeechUtterance 类来实现一段讲话，即话语。它的创建并初始化方法是 AVSpeechUtterance 的 initWithString:方法，其语法形式如下：

```
- (AVSpeechUtterance *)initWithString:(NSString *)string;
```

其中，(NSString *)string 表示要讲话的文本内容，是一个字符串对象。在本实例中就使用了 initWithString:方法创建并初始化了 AVSpeechUtterance 对象，代码如下：

```
AVSpeechUtterance *utterance = [[AVSpeechUtterance alloc] initWithString:
[dict objectForKey:@"content"]];
```

其中，dict objectForKey:@"content"表示要讲话的文本内容。其次，使用 AVSpeechUtterance 类的 voice 属性对讲话的语音进行设置。其语法形式如下：

```
@property(nonatomic, retain) AVSpeechSynthesisVoice *voice ;
```

在本实例中就使用了 AVSpeechUtterance 类的 voice 属性对讲话的语音进行设置，代码如下：

```
utterance.voice = voice;
```

然后，创建一个 AVSpeechSynthesizer 类，它就是一个语言合成器。最后需要使用 AVSpeechSynthesizer 的 speakUtterance:方法将创建的一段讲话放到语音合成器中，形成音频。其语法形式如下：

```
- (void)speakUtterance:(AVSpeechUtterance *)utterance;
```

其中，(AVSpeechUtterance *)utterance 表示一段讲话，即 AVSpeechUtterance 对象。在

本实例中就使用了 `SpeakUtterance` 方法将创建的一段讲话放到语音合成器中。代码如下：

```
[av speakUtterance:utterance];
```

其中，`utterance` 表示一段讲话。

实例 86 播放歌曲的同时显示歌词

【实例描述】

QQ 音乐播放器、千千静听等都可以实现在播放歌曲的时候显示歌词的功能，几乎做到了同步的效果。本实例就为各位读者实现这一功能，这可以使音乐播放器看起来更加实用。运行效果如图 7.7 所示。



图 7.7 运行效果

【实现过程】

- (1) 创建一个项目，命名为“播放歌曲的同时显示歌词”。
- (2) 添加图像 1.jpg、4.png、5.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加歌词文件 2.lrc 到创建项目的 Supporting Files 文件夹中。
- (4) 添加音频文件 3.mp3 到创建项目的 Supporting Files 文件夹中。
- (5) 添加 AVFoundation.framework 到创建的项目中。

(6) 打开 `ViewController.h` 文件，编写代码，实现头文件、对象、插座变量、实例变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface ViewController : UIViewController{
    //对象
    AVAudioPlayer *audioPlayer;
    NSMutableDictionary *LRCDictionary;
    NSMutableArray *musicArray;
    NSMutableArray *timeArray;
```

```
//插座变量
IBOutlet UITableView *lrcTableView;
IBOutlet UIButton *playBtn;

//实例变量
NSInteger lrcLineNumber;
BOOL isPlay;
}
- (IBAction)play:(id) sender;
@end
```

（7）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 7.8 所示。

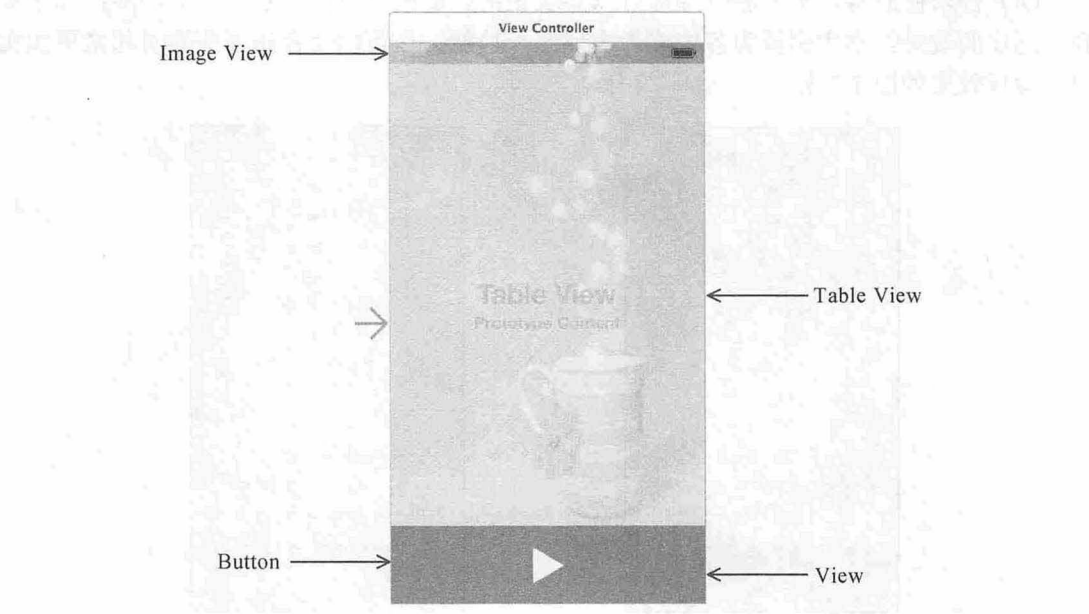


图 7.8 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-6 所示。

表 7-6 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 1.jpg	
Table View	Background: 透明色	dataSource 与 View Controller 关联 delegate 与 View Controller 关联 与插座变量 lrcTableView 关联
Button	Title: （空） Background: 5.png	与插座变量 playBtn 关联 与动作 play:关联
View	Alpha: 0.3 Background: 黑色	

（8）打开 ViewController.m 文件，编写代码，实现在播放歌曲的同时显示歌词。使用的方法如表 7-7 所示。

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewDidLoad

方法在视图加载后调用,实现初始化功能。程序代码如下:

表 7-7 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用,实现初始化
initLRC	初始化歌词
displaySondWord:	动态显示歌词
showTime	0.1 秒一次歌词更新
updateLrcTableView:	动态更新歌词表歌词
tableView:numberOfRowsInSection:	获取行数
tableView:cellForRowAtIndexPath:	获取内容
tableView:heightForRowAtIndexPath:	获取高度
play:	单击按钮,实现音乐的播放

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    lrcLineNumber = 0;
    // Do any additional setup after loading the view, typically from a nib.
    //初始化要加载的曲目
    NSString *path=[[NSBundle mainBundle]pathForResource:@"3" ofType:
    @"mp3"];
    NSURL *url=[NSURL URLWithString:path];
    audioPlayer = [[AVAudioPlayer alloc] initWithContentsOfURL:url error:
    nil];
    //初始化歌词词典
    timeArray = [[NSMutableArray alloc] initWithCapacity:10];
    LRCDictionary = [[NSMutableDictionary alloc] initWithCapacity:10];
    [self initLRC];
    [NSTimer scheduledTimerWithTimeInterval:0.1f target:self selector:
    @selector(showTime) userInfo:nil repeats:YES];           //创建时间定时器
}

```

initLRC 方法实现对歌词的初始化。程序代码如下:

```

- (void)initLRC {
    NSString *LRCPATH = [[NSBundle mainBundle] pathForResource:@"2"
    ofType:@"lrc"];
    NSString *contentStr = [NSString stringWithContentsOfFile:LRCPATH
    encoding:NSUTF8StringEncoding error:nil]; //使用文件的内容创建字符串
    NSArray *array = [contentStr componentsSeparatedByString:@"\n"];
    //循环,为属性添加对象
    for (int i = 0; i < [array count]; i++) {
        NSString *linStr = [array objectAtIndex:i];
        NSArray *lineArray = [linStr componentsSeparatedByString:@" "];
        //判断 lineArray 的第 1 个元素的长度是否大于 8
        if ([lineArray[0] length] > 8) {
            NSString *str1 = [linStr substringWithRange:NSMakeRange(3, 1)];
            NSString *str2 = [linStr substringWithRange:NSMakeRange(6, 1)];
            //判断 str1 的字符串是否为":", str2 的字符串是否为"."
            if ([str1 isEqualToString:@":"] && [str2 isEqualToString:@"."]) {
                NSString *lrcStr = [lineArray objectAtIndex:1];
                NSString *timeStr = [[lineArray objectAtIndex:0] substring-
                WithRange:NSMakeRange(1, 5)];           //分割区间求歌词时间
            }
        }
    }
}

```

```

        [LRCDictionary setObject:lrcStr forKey:timeStr];
        [timeArray addObject:timeStr];          //添加对象
    }
}
}

```

displaySondWord:方法实现动态地显示歌词。程序代码如下:

```

- (void)displaySondWord:(NSUInteger)time {
    for (int i = 0; i < [timeArray count]; i++) {
        NSArray *array = [timeArray[i] componentsSeparatedByString:@":"];
        NSUInteger currentTime = [array[0] intValue] * 60 + [array[1]
        intValue];          //设置当前时间
        if (i == [timeArray count]-1) {
            //求最后一句歌词的时间点
            NSArray *array1 = [timeArray[timeArray.count-1] components
            SeparatedByString:@":"];
            NSUInteger currentTime1 = [array1[0] intValue] * 60 + [array1[1]
            intValue];          //设置当前时间
            //判断 time 是否大于 currentTime1
            if (time > currentTime1) {
                [self updateLrcTableView:i];          //动态更新歌词表歌词
                break;
            }
        } else {
            //求出第一句的时间点,在第一句显示前的时间内一直加载第一句
            NSArray *array2 = [timeArray[0] componentsSeparatedByString:@":"];
            NSUInteger currentTime2 = [array2[0] intValue] * 60 + [array2[1]
            intValue];          //设置当前时间
            if (time < currentTime2) {
                [self updateLrcTableView:0];          //动态更新歌词表歌词
                break;
            }
            //求出下一句的歌词时间点,然后计算区间
            NSArray *array3 = [timeArray[i+1] componentsSeparatedByString:
            @":"];
            NSUInteger currentTime3 = [array3[0] intValue] * 60 + [array3[1]
            intValue];          //设置当前时间
            if (time >= currentTime && time <= currentTime3) {
                [self updateLrcTableView:i];          //动态更新歌词表歌词
                break;
            }
        }
    }
}
}

```

updateLrcTableView:方法实现动态更新歌词表歌词。程序代码如下:

```

- (void)updateLrcTableView:(NSUInteger)lineNumber {
    //重新载入歌词列表 lrcTabView
    lrcLineNumber = lineNumber;
    [lrcTableView reloadData];
    //使被选中的行移到中间
    NSIndexPath *indexPath = [NSIndexPath indexPathForRow:lineNumber
    inSection:0];
    [lrcTableView selectRowAtIndexPath:indexPath animated:YES scrollPosition:
    UITableViewScrollPositionMiddle];
}

```

tableView:cellForRowAtIndexPath:方法实现对内容的获取。程序代码如下:

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath {
    static NSString *cellIdentifier = @"LRCCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
cellIdentifier];
    //判断 cell 是否为空
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:cellIdentifier];
    }
    cell.selectionStyle = UITableViewCellSelectionStyleNone; cell.
backgroundColor = [UIColor clearColor];
    if (indexPath.row == lrcLineNumber) {
        //设置单元格的文本
        cell.textLabel.text = LRCDictionary[timeArray[indexPath.row]];
        //设置文本内容
        cell.textLabel.textColor = [UIColor colorWithRed:0 green:0 blue:0
alpha:1.0];
        cell.textLabel.font = [UIFont systemFontOfSize:15];
    } else {
        //设置单元格的文本
        cell.textLabel.text = LRCDictionary[timeArray[indexPath.row]];
        cell.textLabel.textColor = [UIColor colorWithRed:0 green:0 blue:0
alpha:0.5];
        cell.textLabel.font = [UIFont systemFontOfSize:13];    //设置字体
    }
    cell.textLabel.backgroundColor = [UIColor clearColor]; //设置背景颜色
    cell.textLabel.textAlignment = NSTextAlignmentCenter; //设置对齐方式
    return cell;
}
```

play:方法实现在单击按钮后,实现音频的播放以及暂停等功能。程序代码如下:

```
- (IBAction)play:(id)sender {
    //判断音乐是否播放
    if (isPlay) {
        [audioPlayer play];    //播放音乐
        [playBtn setBackgroundImage:[UIImage imageNamed:@"4.png"] forState:
UIControlStateNormal];
        isPlay = NO;
    } else {
        [audioPlayer pause];    //暂停音乐
        [playBtn setBackgroundImage:[UIImage imageNamed:@"5.png"] forState:
UIControlStateNormal];
        isPlay = YES;
    }
}
```

【代码解析】

本实例的关键功能是歌词的显示和歌词的同步更新功能。下面依次讲解这两个知识点。

1. 歌词的显示

在本实例中歌词的显示,首先,需要将歌词保存在数组中,代码如下:

```
NSString *LRCPATH = [[NSBundle mainBundle] pathForResource:@"2"
ofType:@"lrc"];
NSString *contentStr = [NSString stringWithContentsOfFile:LRCPATH
encoding:NSUTF8StringEncoding error:nil];
NSArray *array = [contentStr componentsSeparatedByString:@"\n"];
```

然后遍历此数组，并将歌词和时间分开，保存在 timeArray 数组中。代码如下：

```
for (int i = 0; i < [array count]; i++) {
    NSString *linStr = [array objectAtIndex:i];
    NSArray *lineArray = [linStr componentsSeparatedByString:@" "];
    if ([lineArray[0] length] > 8) {
        .....
        if ([str1 isEqualToString:@":"] && [str2 isEqualToString:@"."]) {
            NSString *lrcStr = [lineArray objectAtIndex:1];
            NSString *timeStr = [[lineArray objectAtIndex:0]
substringWithRange:NSMakeRange(1, 5)];
            [LRCDictionary setObject:lrcStr forKey:timeStr];
            [timeArray addObject:timeStr];
        }
    }
}
```

最后，将获取的歌词显示在单元格中，代码如下：

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath {
    static NSString *cellIdentifier = @"LRCCell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
cellIdentifier];
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyle-
Default reuseIdentifier:cellIdentifier];
    }
    .....
    cell.textLabel.text = LRCDictionary[timeArray[indexPath.row]];
    cell.textLabel.textColor = [UIColor colorWithRed:0 green:0 blue:0
alpha:0.5];
    cell.textLabel.font = [UIFont systemFontOfSize:13];
    .....
    return cell;
}
```

2. 歌词的同步更新

在本实例中歌词的同步更新，首先需要有一个.lrc 文件，此文件为一种包含“:”形式的、“标签”的、基于纯文本的歌词专用文件，它可以在各类数码播放器中同步显示。它的标准格式如下：

```
[分钟:秒.毫秒] 歌词
```

然后创建一个定时器，代码如下：

```
[NSTimer scheduledTimerWithTimeInterval:0.1f target:self selector:@selector
(showTime) userInfo:nil repeats:YES];
```

其中，定时器会在每隔 0.1 秒执行一次 showTime 方法，此方法会调用 displaySondWord:

方法，该方法实现的功能是动态显示歌词的功能。该方法会遍历保存有歌词时间的数组，然后会判断当前时间是否和 lrc 文件中的歌词时间大致一致，如果一致就更新从而实现同步更新的功能。

实例 87 录音机

【实例描述】

会说话的汤姆猫现在几乎是每一个智能手机中都安装的应用程序，它的主要功能就是对用户进行录音然后自动播放出来。本实例就为各位读者实现会说话的汤姆猫的简易效果，当单击“录音”按钮，进行录音功能，当单击“播放”按钮，进行录制声音的播放。运行效果如图 7.9 所示。



图 7.9 运行效果

【实现过程】

- (1) 创建一个项目，命名为“录音机”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 AVFoundation.framework 到创建的项目中。
- (4) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、对象、实例变量、插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface ViewController : UIViewController<AVAudioPlayerDelegate>{
    //对象
    NSURL *recordedFile;
    AVAudioPlayer *player;
    AVAudioRecorder *recorder;
    BOOL isRecording;
    //插座变量
    IBOutlet UIButton *playButton;
    IBOutlet UIButton *recordButton;
```



```
IBOutlet UIImageView *imageView;
}
//动作
- (IBAction)play:(id)sender;
- (IBAction)record:(id)sender;
@end
```

（5）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 7.10 所示。

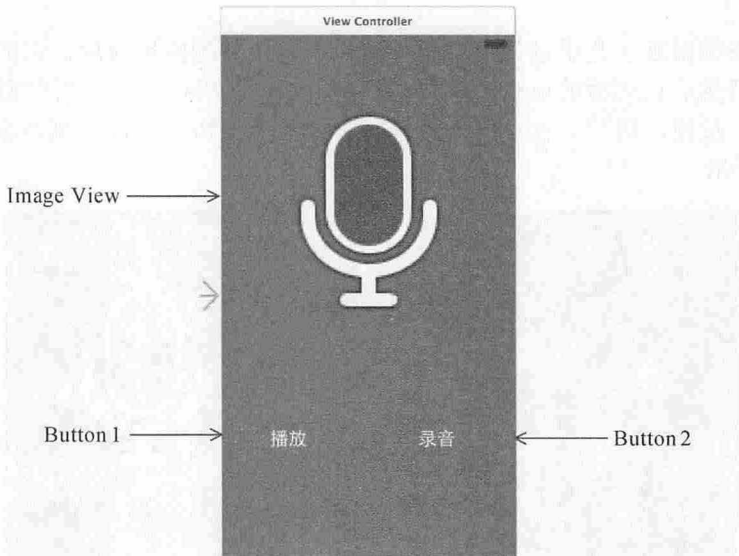


图 7.10 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-8 所示。

表 7-8 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 深灰色	
Image View	Image: 1.png	
Button1	Title: 播放 Font: System 20.0 Text Color: 白色	与插座变量 playButton 关联 与动作 play:关联
Button2	Title: 录音 Font: System 20.0 Text Color: 白色	与插座变量 recordButton 关联 与动作 record:关联

（6）打开 ViewController.m 文件，编写代码，实现录音机的功能即实现录音以及播放录音的功能。使用的方法如表 7-9 所示。

表 7-9 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
play:	实现单击“播放”按钮播放录音
record:	实现单击“录音”按钮进行录音

其中，viewDidLoad 方法在视图加载后调用，实现初始化功能。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    isRecording = NO;
    //设置按钮
    [playButton setEnabled:NO];
    playButton.titleLabel.alpha = 0.5;
    recordedFile = [NSURL fileURLWithPath:[NSTemporaryDirectory()
    stringByAppendingString:@"RecordedFile"]];
    //创建声音会话
    AVAudioSession *session = [AVAudioSession sharedInstance];
    NSError *sessionError;
    [session setCategory:AVAudioSessionCategoryPlayAndRecord error:
    &sessionError];
    //判断删除“窗户的”声音会话对象是否正确
    if(session == nil){
        NSLog(@"Error creating session: %@", [sessionError description]);
        //输出
    } else{
        [session setActive:YES error:nil];
    }
    imageView.hidden=YES; //隐藏图像视图
}
```

play:方法实现单击“播放”按钮，播放录制的音频。程序代码如下：

```
- (IBAction)play:(id)sender {
    //判断是否正在播放音频
    if([player isPlaying]){
        //暂停播放
        [player pause];
        [playButton setTitle:@"播放" forState:UIControlStateNormal];
    }else{
        //播放
        [player play];
        [playButton setTitle:@"STOP" forState:UIControlStateNormal];
    }
}
```

record:实现单击“录音”按钮，实现录音效果。程序代码如下：

```
- (IBAction)record:(id)sender {
    imageView.hidden=NO;
    //判断是否正在录音
    if(!isRecording){
        //录音
        isRecording = YES;
        [recordButton setTitle:@"STOP" forState:UIControlStateNormal];
        //设置标题
        [playButton setEnabled:NO];
        [playButton.titleLabel setAlpha:0.5]; //设置透明度
        recorder = [[AVAudioRecorder alloc] initWithURL:recordedFile
        settings:nil error:nil];
        [recorder prepareToRecord]; //准备录音
        [recorder record];
        player = nil;
    }
}
```

```

imageView.hidden=NO; //显示图像视图
}else{
//停止录音
isRecording = NO;
[recordButton setTitle:@"录音" forState:UIControlStateNormal];
//设置标题
[playButton setEnabled:YES];
[playButton.titleLabel setAlpha:1]; //设置透明度
[recorder stop];
recorder = nil;
NSError *playerError;
player = [[AVAudioPlayer alloc] initWithContentsOfURL:recordedFile
error:&playerError];
//判断是否出错
if (player == nil)
{
    NSLog(@"Error creating player: %@", [playerError description]);
    //输出
}
player.delegate = self;
imageView.hidden=YES; //隐藏图像视图
}
}

```

【代码解析】

本实例关键功能是在音频的录制。下面就是这个知识点的详细讲解。

AVFoundation.framework 框架中的 AVAudioRecorder 类可以实现录制音频。它的创建初始化需要使用 AVAudioRecorder 的 initWithURL:settings:error:方法实现。其语法形式如下：

```

- (id)initWithURL:(NSURL *)url settings:(NSDictionary *)settings error:
(NSError **)outError;

```

其中，(NSURL *)url 表示录制文件的系统位置；(NSDictionary *)settings 表示用于记录会话设置；(NSError **)outError 表示错误。在本实例中就使用了 initWithURL:settings:error:方法创建并初始化 AVAudioRecorder 对象，其代码如下：

```

recorder = [[AVAudioRecorder alloc] initWithURL:recordedFile settings:nil
error:nil];

```

其中，recordedFile 表示录制文件的系统位置；nil 表示没有用于记录会话设置；nil 表示没有错误返回。

实例 88 获取系统中所有的音频文件

【实例描述】

在 iOS 系统中很多的音频文件。但是没有很少有人知道到底有哪些音频文件。本实例就为各位读者将 iOS 系统中的所有音频文件列举在表视图中，并且在单击某一行的内容后，就会进行播放。运行效果如图 7.11 所示。



图 7.11 运行效果

【实现过程】

- (1) 创建一个项目，命名为“获取系统中所有的音频文件”。
- (2) 添加 AudioToolbox.framework 到创建的项目中。
- (3) 打开 ViewController.h 文件，编写代码，实现头文件以及对象的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AudioToolbox/AudioToolbox.h>
@interface ViewController : UIViewController{
    NSMutableArray *audioFileList; //声明可变量组
}
@end
```

- (4) 打开 Main.storyboard 文件，从视图库中添加 Table View 表视图到设计界面中，将此视图的 dataSource、delegate 与 View Controller 关联。
- (5) 打开 ViewController.m 文件，编写代码，实现音频文件的获取。使用的方法如表 7-10 所示。

表 7-10 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
loadAudioFileList	加载音频文件列表
numberOfSectionsInTableView:	获取块数
tableView:numberOfRowsInSection:	获取行数
tableView:cellForRowAtIndexPath:	获取内容
tableView:didSelectRowAtIndexPath:	选择行的响应

loadAudioFileList 方法实现音频文件列表的加载。程序代码如下：

```
-(void)loadAudioFileList{
    audioFileList = [[NSMutableArray alloc] init];
    NSFileManager *fileManager = [[NSFileManager alloc] init];
```

```

//创建文件管理器
NSURL *directoryURL = [NSURL URLWithString:@"~/System/Library/Audio"];
NSArray *keys = [NSArray arrayWithObject:NSURLIsDirectoryKey];
//创建数组

//枚举目录中的内容
NSDirectoryEnumerator *enumerator = [fileManager enumeratorAtURL:
directoryURL includingPropertiesForKeys:keys options:0 errorHandler:^(NSURL
*url, NSError *error) {return YES;}}];
//遍历
for (NSURL *url in enumerator) {
    NSError *error;
    NSNumber *isDirectory = nil;
    if (![url getResourceValue:&isDirectory forKey:NSURLIsDirectoryKey
error:&error]) {
    }
    //判断 isDirectory 的布尔值
    else if (![isDirectory boolValue]) {
        [audioFileList addObject:url]; //添加对象
    }
}
}

```

tableView:cellForRowAtIndexPath:方法实现获取表视图中的内容。程序代码如下:

```

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"Cell";
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:
CellIdentifier];
    //判断 cell 是否为空
    if (cell == nil) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyle-
Default reuseIdentifier:CellIdentifier];
    }
    cell.textLabel.text = [[audioFileList objectAtIndex:indexPath.row]
lastPathComponent]; //设置文本内容
    return cell;
}

```

tableView:didSelectRowAtIndexPath:方法实现播放选中的行。程序代码如下:

```

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    SystemSoundID soundID;
    AudioServicesCreateSystemSoundID((__bridge_retained
CFURLRef) [audioFileList objectAtIndex:indexPath.row], &soundID);
    //创建系统声音
    AudioServicesPlaySystemSound(soundID); //播放系统声音
}

```

【代码解析】

本实例关键功能是音频文件的获取。下面就是这个知识点的详细讲解。

NSFileManager 的 enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:方法可以实现指定目录中的枚举。其语法形式如下:

```

- (NSDirectoryEnumerator *)enumeratorAtURL:(NSURL *)url including
PropertiesForKeys:(NSArray *)keys options:(NSDirectoryEnumerationOptions)

```



```
mask errorHandler:(BOOL (^)(NSURL *url, NSError *error))handler;
```

其中，参数说明如下：

- ❑ (NSURL *)url 表示枚举位置的目录；
- ❑ (NSArray *)keys 是键数组，表示性能识别，就是在枚举中获取的每一项；
- ❑ (NSDirectoryEnumerationOptions)mask 表示枚举选项，一个有效的选项列表；
- ❑ (BOOL (^)(NSURL *url, NSError *error))handler 在错误发生时调用。

在本实例中就是使用了 `enumeratorAtURL:includingPropertiesForKeys:options:errorHandler:` 方法实现了将指定目录中的文件枚举出来，代码如下：

```
NSDirectoryEnumerator *enumerator = [fileManager enumeratorAtURL:
directoryURL includingPropertiesForKeys:keys options:0 errorHandler:^(
NSURL *url, NSError *error) {return YES;}}];
```

其中，参数说明如下：

- ❑ directoryURL 表示枚举位置的目录；
- ❑ keys 是键数组，表示性能识别，就是在枚举中获取的每一项；
- ❑ 0 表示没有枚举选项；
- ❑ {return YES;} 表示在错误发生时调用。

实例 89 讯飞识别

【实例描述】

讯飞输入法在很多的智能手机中都有安装，它独家推出方言语音输入，支持粤语、四川话、河南话、东北话方言识别，开启语音识别新时代。它也是支持 iOS 7 越狱的中文输入法。本实例就为读者实现讯飞输入法的语言识别部分。当单击“识别”按钮，用户可以进行说话，并在标签中显示说话的内容，当单击“播放”按钮，在标签中的内容就会读出。运行效果如图 7.12 所示。

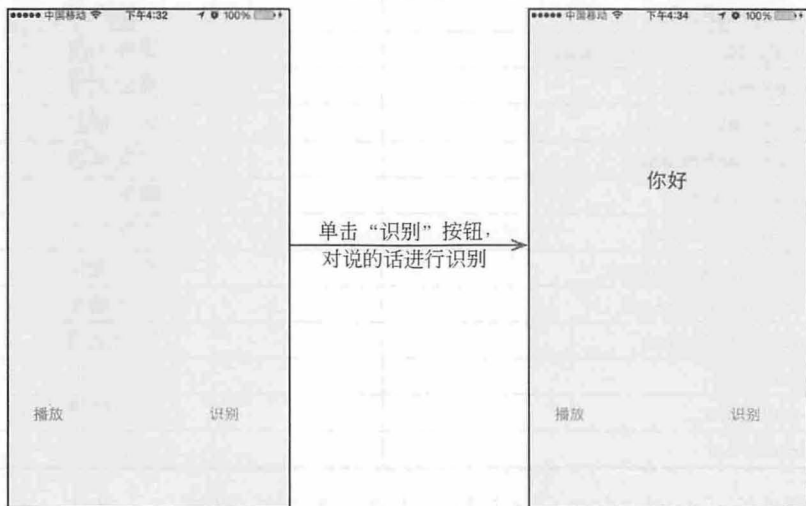


图 7.12 运行效果

【实现过程】

(1) 创建一个项目，命名为“讯飞识别”。

(2) 添加 AudioToolbox.framework、SystemConfiguration.framework、libz.dylib、AdSupport.framework、iflyMSC.framework 到创建项目的 Frameworks 文件夹中。

(3) 创建一个基于 NSObject 类的 Xunfei 类。

(4) 打开 Xunfei.h 文件，编写代码，实现头文件、遵守协议、对象、属性以及方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
//头文件
#import "iflyMSC/IFlySpeechSynthesizer.h"
#import "iflyMSC/IFlySpeechRecognizer.h"
@interface Xunfei : NSObject<IFlySpeechSynthesizerDelegate,
IFlySpeechRecognizerDelegate>{
    //对象
    IFlySpeechSynthesizer* iFlySpeechSynthesizer;
    IFlySpeechRecognizer*iflySpeechRecognizer;
    NSArray *array;
}
@property(nonatomic, copy) void(^onResult) (NSString*);
//方法
+ (id)shareManager;
- (void)playVoice: (NSString*) str;
- (void)discernBlock: (void(^)(NSString*)) a;
@end
```

(5) 打开 Xunfei.m 文件，编写代码，实现播放和识别语音。使用的方法如表 7-11 所示。

表 7-11 Xunfei.m文件中方法总结

方 法	功 能
shareManager	获取单例
playVoice:	播放语音
discernBlock:	识别语言
onSpeakBegin	开始合成
onBufferProgress:message:	缓冲进度
onSpeakProgress:	播放进度
onSpeakPaused	暂停播放
onSpeakResumed	恢复播放
onCompleted:	结束
onSpeakCancel	正在取消
onVolumeChanged:	音量变化
onBeginOfSpeech	开始录音
onEndOfSpeech	停止录音
onError:	发生错误
onResults:	识别结果
onCancel	取消识别

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，playVoice:方法实现语音的播放功能。程序代码如下：

```

-(void)playVoice:(NSString*)str{
    iFlySpeechSynthesizer = [IFlySpeechSynthesizer createWithParams:
        @"appid=52bbb432" delegate:self];
    iFlySpeechSynthesizer.delegate = self;
    // 设置语音合成的参数
    [iFlySpeechSynthesizer setParameter:@"speed" value:@"50"];
    //合成的语速,取值范围 0~100
    [iFlySpeechSynthesizer setParameter:@"volume" value:@"50"];
    //合成的音量,取值范围 0~100
    //发音人,默认为“xiaoyan”;可设置的参数列表可参考个性化发音人列表
    [iFlySpeechSynthesizer setParameter:@"voice_name" value:@"xiaoyan"];
    //音频采样率,目前支持的采样率有 16000 和 8000
    [iFlySpeechSynthesizer setParameter:@"sample_rate" value:@"8000"];
    [iFlySpeechSynthesizer startSpeaking:str];
}

```

discernBlock:方法实现语音的识别功能,程序代码如下:

```

-(void)discernBlock:(void(^)(NSString*))a{
    self.onResult=a;
    iflySpeechRecognizer = [IFlySpeechRecognizer createRecognizer:
        @"appid=52bbb432,timeout=20000" delegate:self]; //时间按照毫秒计算
    iflySpeechRecognizer.delegate = self; //设置代理
    [iflySpeechRecognizer setParameter:@"domain" value:@"sms"];
    [iflySpeechRecognizer setParameter:@"sample_rate" value:@"16000"];
    [iflySpeechRecognizer setParameter:@"plain_result" value:@"0"];
    [iflySpeechRecognizer setParameter:@"vad_eos" value:@"1800"];
    //设置多少毫秒后断开录音
    BOOL ret = [iflySpeechRecognizer startListening];
}

```

onResults:方法实现识别结果。程序代码如下:

```

- (void) onResults:(NSArray *) results{
    NSMutableString *result =
    [[NSMutableString alloc] init];
    NSDictionary *dic = [results objectAtIndex:0];
    //遍历
    for (NSString *key in dic) {
        [result appendFormat:@"%s",key];
    }
    self.onResult(result);
}

```

(6) 打开 ViewController 文件,编写代码,实现头文件、插座变量以及动作的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "Xunfei.h"
@interface ViewController : UIViewController{
    IBOutlet UILabel *label;
}
//动作
- (IBAction)play:(id)sender;
- (IBAction)check:(id)sender;
@end

```

(7) 打开 Main.storyboard 文件,对 View Controller 视图控制器的设计界面进行设计,效果如图 7.13 所示。

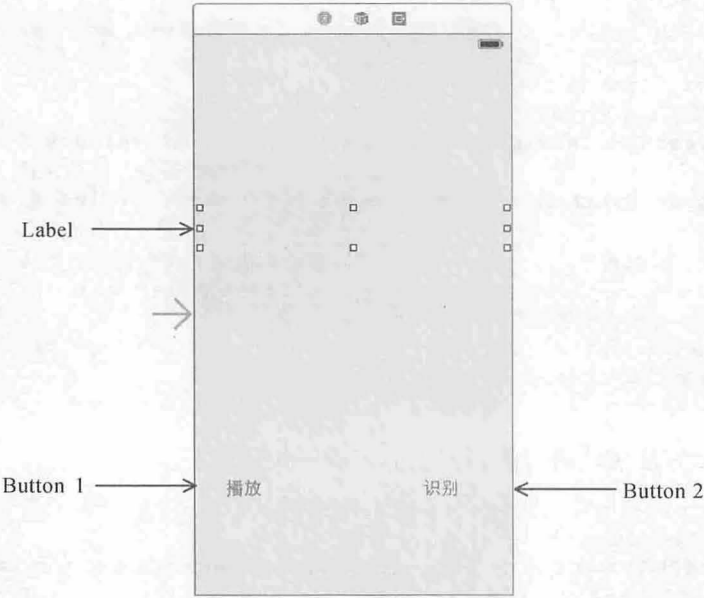


图 7.13 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-12 所示。

表 7-12 视图、控件设置

视图、控件	属 性 设 置	其 他
Label	Text: (空) Font: System 23.0 Alignment: 居中	
Button1	Title: 播放 Font: System 18.0	与动作 play:关联
Button2	Title: 识别 Font: System 18.0	与动作 check:关联

(8) 打开 ViewController.m 文件，编写代码，实现单击按钮后，识别以及播放语音。程序代码如下：

```
//单击“播放”按钮，实现语音的播放
- (IBAction)play:(id)sender {
    Xunfei *manager=[Xunfei shareManager];
    [manager playVoice:[NSString stringWithFormat:@"%s",label.text]];
}
//单击“识别”按钮，实现语言的识别
- (IBAction)check:(id)sender {
    Xunfei *manager=[Xunfei shareManager];
    [manager discernBlock:^(NSString *str) {
        //判断 str 的长度是否大于 1
        if (str.length>1) {
            label.text=str;
            //设置文本内容
        }
    }];
}
```

【代码解析】

本实例关键功能是声音的识别以及播放功能。下面依次讲解这两个知识点。

1. 声音的识别

讯飞声音的识别需要使用它自己的 `IFlySpeechRecognizer` 类进行识别，它的创建需要使用它自带的 `createRecognizer:delegate:` 方法。在本实例中创建 `IFlySpeechRecognizer` 就使用了此方法，代码如下：

```
iflySpeechRecognizer = [IFlySpeechRecognizer createRecognizer:
@"appid=52bbb432,timeout=20000" delegate:self];
```

其中，`@ "appid=52bbb432,timeout=20000"` 表示初始化实现的参数；`self` 表示委托对象。

2. 播放

讯飞声音的播放也用它自带的 `IFlySpeechSynthesizer` 类。它的创建需要使用它自带的 `createWithParams:delegate:` 方法。在本实例中创建 `IFlySpeechSynthesizer` 就使用了此方法，代码如下：

```
iFlySpeechSynthesizer = [IFlySpeechSynthesizer createWithParams:
@"appid=52bbb432" delegate:self];
```

其中，`@ "appid=52bbb432"` 表示初始化实现的参数；`self` 表示委托对象。

实例 90 音乐播放器

【实例描述】

本实例实现的是一个自制的音乐播放器。此音乐播放器几乎完成了和 QQ 播放器相同的功能。运行效果如图 7.14 所示。

【实现过程】

- (1) 创建一个项目，命名为“音乐播放器”。
- (2) 添加 `AVFoundation.framework` 到创建的项目中。
- (3) 添加音频文件“等风的日子.mp3”、“奇迹.mp3”、`God only knows.mp3`、`Ring.mp3` 到创建项目的 Supporting Files 文件夹中。
- (4) 添加图像 `background.png`、`laba.png`、`pugongying.png`、`aboveMusic.png`、`menu.png`、`nextMusic.png`、`pause.png`、`play.png` 到创建项目的 Supporting Files 文件夹中。
- (5) 打开 `ViewController.h` 文件，编写代码，实现头文件、遵守协议、对象、实例变量、插座变量、动作以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface ViewController : UIViewController<AVAudioPlayerDelegate>{
    //对象
    AVAudioPlayer* _audioPlayer;
    NSMutableArray* _pNames;
    NSMutableArray* _hNames;
```



```

NSTimer* processTimer;
UISlider* _volumeSlider;
NSTimer* timer1;
int _songIndex;
//插座变量
IBOutlet UILabel *_label;           //声明关于标签的插座变量
IBOutlet UISlider *_slider;
IBOutlet UIButton *button;          //声明关于按钮的插座变量
IBOutlet UIButton *menubutton;
IBOutlet UITableView *tableView;    //声明关于表视图的插座变量
}
//动作
- (IBAction)prior:(id) sender;       //上一首
- (IBAction)next:(id) sender;
- (IBAction)play:(id) sender;
- (IBAction)showVolume:(id) sender; //显示音量
- (IBAction)showHide:(id) sender;
- (void)loadMusic:(NSString*)name type:(NSString*)type; //加载音乐
@end

```

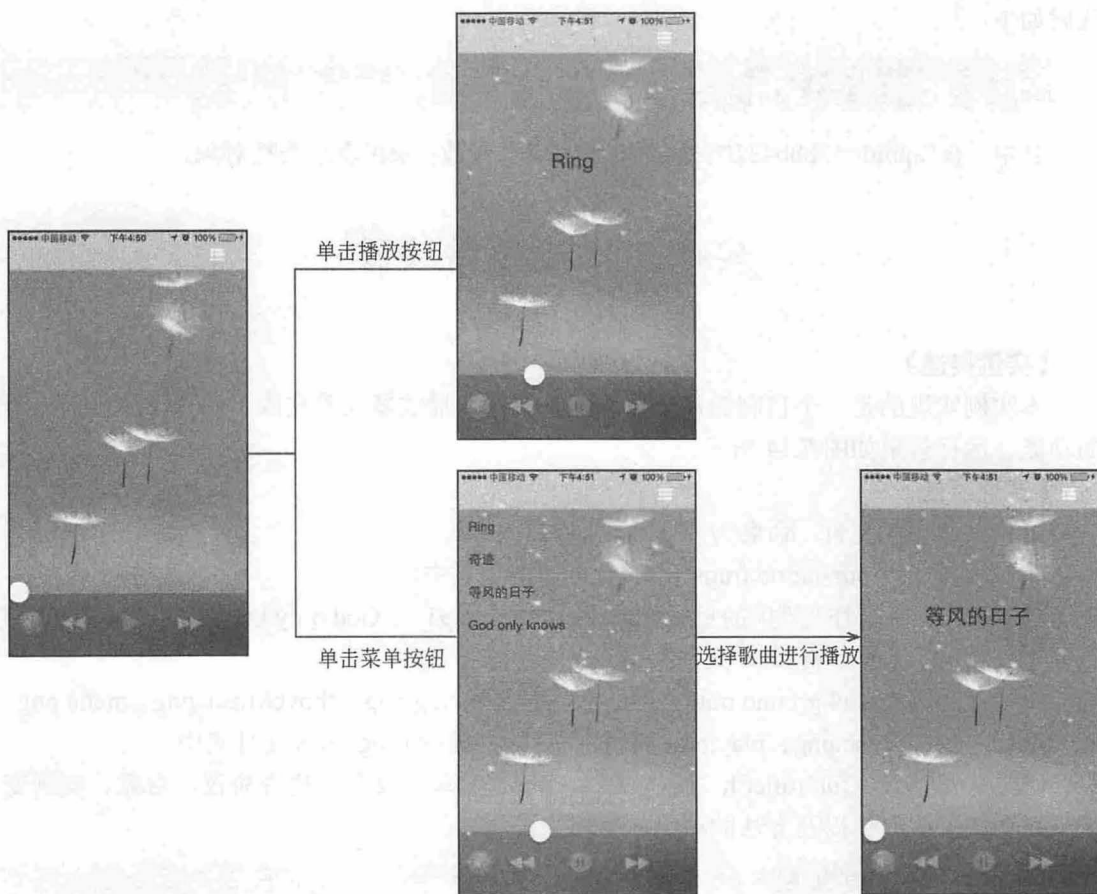


图 7.14 运行效果

(6) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 7.15 所示。

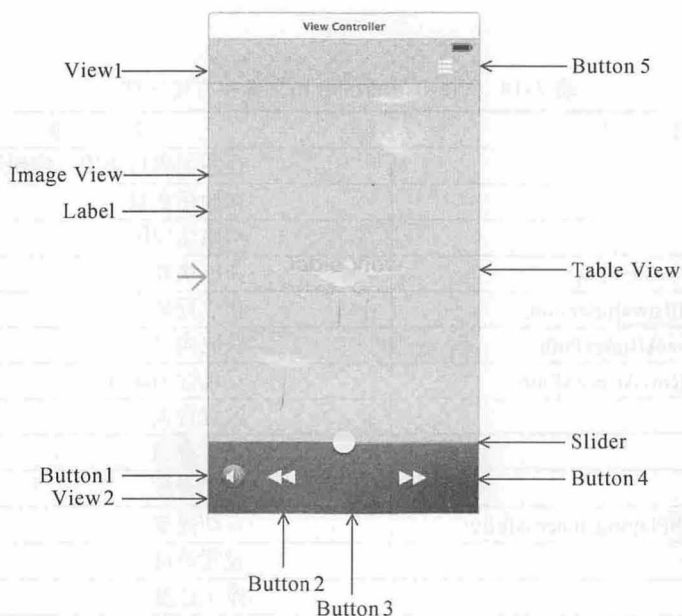


图 7.15 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-13 所示。

表 7-13 视图、控件设置

视图、控件	属 性 设 置	其 他
View1	Background: 浅蓝色	
Image View	Image: background.png	
Label	Text: (空) Font: Helvetica Neue 28.0 Alignment: 居中	与插座变量_label 关联
Button1	Title: (空) Background: laba.png	与动作 showVolume:关联
View2	Alpha: 0.25 Background: 黑色	
Button2	Title: (空) Background: aboveMusic.png	与动作 prior:关联
Button3	Title: (空)	与插座变量 button 关联 与动作 play:关联
Button4	Title: (空) Background: nextMusic.png	与动作 next:关联
Slider		
Table View		与插座变量 tView 关联 dataSource 与 View Controller 关联 delegate 与 View Controller 关联
Button5	Title: (空) Background: menu.png	与插座变量 menubutton 关联 与动作 showHide:关联

(7) 打开 ViewController.m 文件, 编写代码, 实现音乐播放器的功能。使用的方法如

表 7-14 所示。

表 7-14 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
dataInit	初始化数据
viewInit	初始化视图
numberOfSectionsInTableView:	获取块数
tableView:numberOfRowsInSection:	获取行数
tableView:cellForRowAtIndexPath:	获取内容
tableView:didSelectRowAtIndexPath:	实现行的选择
volumeSet:	设置音量
showVolume	显示音量
hideVolume	隐藏音量
audioPlayerDidFinishPlaying:successfully:	自动播放
processSet:	设置进度
processTimerStop	停止进度
process	歌曲进度
loadMusic:type:	加载音乐
pugongying	动画
onAnimationComplete:finished:	动画完成
prior:	上一首
next:	下一首
play:	播放

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewInit 方法实现对视图的初始化功能。程序代码如下：

```
-(void) viewInit
{
    //对滑块控件的设置
    _slider.maximumValue=100; //设置滑块控件的最大值
    _slider.minimumValue=0; //设置滑块控件的最小值
    _slider.value=0;
    _slider.continuous=NO;
    [_slider addTarget:self action:@selector(processSet:) forControlEvents:
    UIControlEventValueChanged]; //添加动作
    //创建定时器
    processTimer=[NSTimer scheduledTimerWithTimeInterval:0.1 target:self
    selector:@selector(process) userInfo:nil repeats:YES];
    [self loadMusic:[_pNames objectAtIndex:0] type:@"mp3"];//加载歌曲
    //创建并设置滑块控件对象，用来调节音量
    _volumeSlider= [[UISlider alloc] initWithFrame:CGRectMake(-95, 350, 220,
    5)];
    _volumeSlider.maximumValue=1;
    _volumeSlider.minimumValue=0;
    _volumeSlider.value= 0.5; //设置当前值
    _volumeSlider.hidden=YES;
    [_volumeSlider addTarget:self action:@selector(volumeSet:) forControlEvents:
    UIControlEventValueChanged]; //添加动作
    _volumeSlider.transform= CGAffineTransformMakeRotation(-90* M_PI/180);
```

```
[self.view addSubview:_volumeSlider];
}
```

tableView:didSelectRowAtIndexPath:方法实现选择表视图的某一行后,播放此行的歌曲内容。程序代码如下:

```
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
(NSIndexPath *)indexPath{
    NSString *aa=[_pNames objectAtIndex:indexPath.row];
    NSString *path=[[NSBundle mainBundle]pathForResource:aa ofType:@"mp3"];
    NSURL *url=[NSURL URLWithString:path];
    //创建并设置 AVAudioPlayer 音频播放器对象
    _audioPlayer=[[AVAudioPlayer alloc]initWithContentsOfURL:url error:nil];
    [_audioPlayer play]; //播放歌曲
    tableView.hidden=YES;
    _label.hidden=NO; //隐藏标签
    _label.text=aa;
    //创建定时器对象
    timer1=[NSTimer scheduledTimerWithTimeInterval:0.1 target:self selector:
    @selector(pugongying) userInfo:nil repeats:YES];
}
```

audioPlayerDidFinishPlaying:successfully:方法实现在播放完一首歌曲后,自动进入下一首歌曲的播放。程序代码如下:

```
-(void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player successfully:
(BOOL)flag
{
    _songIndex++;
    if(_songIndex==_pNames.count)
        _songIndex= 0;
    [self loadMusic:[_pNames objectAtIndex:_songIndex] type:@"mp3"]; //加载歌曲
    _label.text= [_hNames objectAtIndex:_songIndex]; //设置文本内容
    [_audioPlayer play];
}
```

loadMusic:type:方法实现音频文件加载。程序代码如下:

```
-(void)loadMusic:(NSString*)name type:(NSString*)type
{
    NSString* path= [[NSBundle mainBundle] pathForResource: name ofType:
    type];
    NSURL* url = [NSURL URLWithString:path];
    _audioPlayer= [[AVAudioPlayer alloc] initWithContentsOfURL:url error:
    nil];
    _audioPlayer.delegate=self; //设置委托
    _audioPlayer.volume= 0.5; //设置音量
    [_audioPlayer prepareToPlay];
}
```

pugongying 方法实现的是一个特效,在播放歌曲的同时飘一些蒲公英的动画效果。程序代码如下:

```
-(void)pugongying
{
    //生成随机数
    int startX= random()%320;
    int endX= random()%320;
```

```

int width= random()%25;
CGFloat time= (random()%100)/10+5;
CGFloat alp= (random()%9)/10.0+0.1;
UIImage* image= [UIImage imageNamed:@"pugongying.png"];    //创建图像
UIImageView* imageView = [[UIImageView alloc] initWithImage:image];
imageView.frame= CGRectMake(startX,-1*width,width,width);
    //设置框架
imageView.alpha=alp;    //设置透明度
[self.view addSubview:imageView];
//飘落蒲公英的动画效果
[UIView beginAnimations:nil context:((__bridge void *) (imageView))];
[UIView setAnimationDuration:time];    //设置动画持续时间
if (endX>50&&endX<270)
{
    imageView.frame= CGRectMake(endX, 270-width/2, width, width);
    //设置框架
}
else if((endX>10&&endX<50)|| (endX>270&&endX<310))
    imageView.frame= CGRectMake(endX, 400-width/2, width, width);
    //设置框架
else
    imageView.frame= CGRectMake(endX, 480, width, width);    //设置框架
[UIView setAnimationDidStopSelector:@selector(onAnimationComplete:
finished:context:)];
[UIView setAnimationDelegate:self];
[UIView commitAnimations];
}

```

prior:方法实现单击上一曲按钮，会自动播放当前歌曲的前一首歌曲。程序代码如下：

```

- (IBAction)prior:(id)sender {
    BOOL playFlag;
    //判断音乐是否正在播放
    if(_audioPlayer.playing){
        playFlag=YES;
        [_audioPlayer stop];    //停止音乐的播放
    }else{
        playFlag=NO;
    }
    _songIndex--;
    //判断_songIndex 是否小于 0
    if(_songIndex<0)
        _songIndex= _pNames.count-1;
    [self loadMusic:[_pNames objectAtIndex:_songIndex] type:@"mp3"];
    //加载音乐
    _label.text= [_hNames objectAtIndex:_songIndex];    //设置标签的文本内容
    if(playFlag==YES){
        [_audioPlayer play];    //播放音乐
    }
}

```

play:方法实现单击播放按钮进行歌曲的播放，此时的播放按钮会变为暂停按钮，单击此暂停按钮，播放的歌曲会停止播放。程序代码如下：

```

- (IBAction)play:(id)sender {
    //判断音乐是否正在播放
    if(_audioPlayer.playing) {
        [button setImage:[UIImage
imageNamed:@"play.png"] forState:UIControlStateNormal];
    }
}

```



```

        [_audioPlayer pause]; //让音乐暂停
        [timer1 invalidate]; //让定时器失效
    } else {
        [self loadMusic:[_pNames objectAtIndex:_songIndex] type:@"mp3"];
        _label.text= [_hNames objectAtIndex:_songIndex]; //设置文本内容
        [button setImage:[UIImage imageNamed:@"pause.png"] forState:
        UIControlStateNormal];
        timer1=[NSTimer scheduledTimerWithTimeInterval:0.1 target:self
        selector:@selector(pugongying) userInfo:nil repeats:YES];
        //创建时间定时器
        [_audioPlayer play];
    }
}

```

【代码解析】

本实例的关键功能是调整音乐播放的进度、音量的调整功能。下面依次讲解这两个知识点。

1. 调整音乐播放的进度

在每一个播放器中，都可以实现调节视频或者是音乐的播放进度，其中，音频的播放进度需要使用 AVAudioPlayer 的 currentTime 属性进行设置，此属性的功能是获取音频播放时间的播放点。其语法形式如下：

```
@property NSTimeInterval currentTime;
```

在本实例就是使用了 currentTime 属性实现了音乐播放进度的调整，代码如下：

```
_audioPlayer.currentTime=slider.value/100*_audioPlayer.duration;
```

2. 音量的调整

AVAudioPlayer 的 volume 属性可以对音频的音量进行设置。其语法形式如下：

```
@property float volume;
```

在本实例中就使用了 volume 属性实现了对音量的调整功能。代码如下：

```
audioPlayer.volume= slider.value;
```

实例 91 视频播放器

【实例描述】

当今人们使用手机排解无聊最有效的方法就是听音乐或者看视频等。本实例就为各位读者实现一个播放本地视频的视频播放器的功能。单击“播放”按钮视频开始播放，单击“停止”按钮，视频停止播放。运行效果如图 7.16 所示。

【实现过程】

- (1) 创建一个项目，命名为“视频播放器”。
- (2) 添加视频 1.mp4 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 MediaPlayer.framework 到创建的项目中。
- (4) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量、对象以及动作

的声明。程序代码如下：



图 7.16 运行效果

```
#import <UIKit/UIKit.h>
#import <MediaPlayer/MediaPlayer.h>
@interface ViewController : UIViewController{
    IBOutlet UIView *vv;
    MPMoviePlayerController *movie;
}
//动作
- (IBAction)play:(id)sender;
- (IBAction)pause:(id)sender;
@end
```

//声明插座变量
//声明对象

(5) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 7.17 所示。

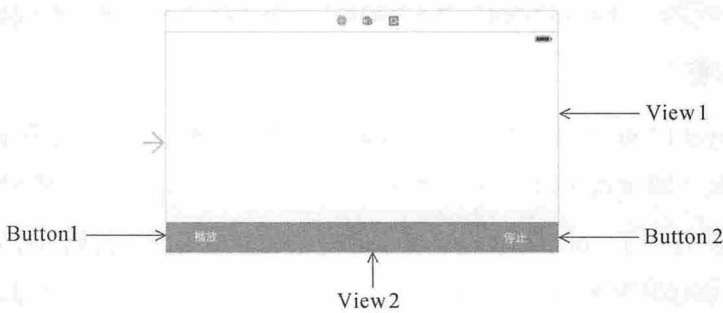


图 7.17 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 7-15 所示。

表 7-15 视图、控件设置

视图、控件	属 性 设 置	其 他
View1		与插座变量 vv 关联
View2	Background: 灰色	
Button1	Text: 播放 Font: System 17.0 Text Color: 白色	与动作 play:关联
Button2	Text: 停止 Font: System 17.0 Text Color: 白色	与动作 pause:关联

(6) 打开 ViewController.m 文件, 编写代码, 实现视频播放器的功能。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    NSString *file=[[NSBundle mainBundle]pathForResource:@"1" ofType:
@"mp4"];
    NSURL *url=[NSURL URLWithString:url];
    movie=[[MPMoviePlayerController alloc] initWithContentURL:url];
    //创建对象
    [movie.view setFrame:vv.frame];           //设置播放的框架
    [self.view addSubview:movie.view];
}
//单击“播放”按钮, 播放视频
- (IBAction)play:(id)sender {
    [movie play];                             //播放视频
}
//单击“停止”按钮, 播放停止
- (IBAction)pause:(id)sender {
    [movie stop];                             //播放停止
}
```

【代码解析】

本实例关键功能是视频的播放。下面就是这个知识点的详细讲解。

基本的视频播放一般可以使用 MPMoviePlayerController 类实现。它的实现, 首先需要设置好播放视频的路径, 接着创建并使用 URL 去初始化 MPMoviePlayerController 对象, 在本实例中的代码如下:

```
movie=[[MPMoviePlayerController alloc] initWithContentURL:url];
```

其次, 需要对视频播放的范围进行设置, 在本实例中使用了 setFrame:方法, 代码如下:

```
[movie.view setFrame:vv.frame];
```

最后使用 MPMoviePlayerController 的 play 方法实现视频的播放, 在本实例中的代码如下:

```
[movie play];
```

第 8 章 内置的应用程序

在 iOS 中有很多内置的应用程序，例如日历、地址簿、短信等。开发者可以在自己的应用程序中直接使用这些内置的应用程序，从而避免功能的重复实现。本章将使用这些内置的应用程序开发一些实例。

实例 92 工作日计算器

【实例描述】

日历提供了查看日期以及节假日的放假安排等功能。如果想要查看本月上班的天数，或者是一段时间的上班天数，盯着日历一天一天地数太麻烦了。本实例就针对这个缺点实现一个工作日计算器的功能。当用户在文本框中输入开始时间以及结束后（注意开始时间和结束时间不可以是节假日），单击“计算”按钮，就会显示出此段时间的工作天数。运行效果如图 8.1 所示。



图 8.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“工作日计算器”。
- (2) 打开 ViewController.h 文件，编写代码，实现插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    //插座变量
    IBOutlet UITextField *tf1;
```

```
IBOutlet UITextField *tf2;
IBOutlet UIView *vv;
IBOutlet UIDatePicker *datePicker;
IBOutlet UILabel *dayLabel;
}
//动作
- (IBAction)select:(id)sender;
- (IBAction)slect1:(id)sender;
- (IBAction)selectdate:(id)sender;           //实现日期选择器的选择
- (IBAction)jishua:(id)sender;              //计算
- (IBAction)finsh:(id)sender;
@end
```

(3) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 8.2 所示。

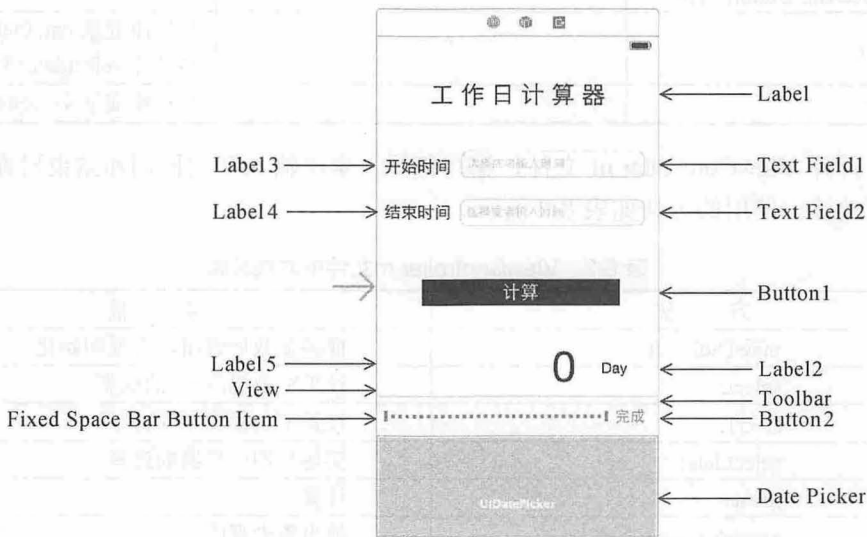


图 8.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 8-1 所示。

表 8-1 视图、控件设置

视图、控件	属 性 设 置	其 他
Label1	Text: 工作日计算器 Font: Helvetica Neue 27.0 Alignment: 居中	
Label2	Text: Day Alignment: 居中	
Label3	Text: 开始时间 Font: System 19.0	
Label4	Text: 结束时间 Font: System 19.0	
Label5	Text: 0 Font: System 19.0 Alignment: 居中	与插座变量 dayLabel 关联

续表

视图、控件	属 性 设 置	其 他
Text Field1	Placeholder: 选择或者输入时间	与插座变量 tf1 关联 与动作 select:关联
Text Field2	Placeholder: 选择或者输入时间	与插座变量 tf2 关联 与动作 select1:关联
Button1	Title: 计算 Font: System 21.0 Text Color: 白色 Background: 黑色	与动作 jishua:关联
Button2	Title: 完成	与动作 finsh:关联
Toolbar	Background: 黑色	
Fixed Space Bar Button Item		
Date Picker		与插座变量 datePicker 关联 与动作 selectdate:关联
View		与插座变量 vv 关联

(4) 打开 ViewController.m 文件, 编写代码, 实现输入开始日期和结束日期后, 计算工作日的功能。使用的方法如表 8-2 所示。

表 8-2 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
select:	设置空白视图 vv 的框架
slect1:	设置空白视图 vv 的框架
selectdate:	实现日期选择器的选择
jishua:	计算
alterMessage:	弹出警告视图
dateToStringDate:	将日期转换为字符串
dateFromString:	将字符串转换为日期
finsh:	单击“完成”按钮, 关闭日期选择器

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中, selectdate:方法实现在日期选择器中选择日期, 并显示在对应的文本框中。程序代码如下:

```
- (IBAction)selectdate:(id)sender {
    if(tf1.resignFirstResponder){
        tf1.text=[self dateToStringDate:datePicker.date];    //设置文本内容
    }else if(tf2.resignFirstResponder){
        tf2.text=[self dateToStringDate:datePicker.date];    //设置文本内容
    }
}
```

jishua:方法通过获取开始时间和结束时间中的日期进行计算, 从而得到工作日的天数。程序代码如下:

```
- (IBAction)jishua:(id)sender {
    //开始时间和结束时间中的字符串, 并将字符串转换为日期
    NSDate *beginDate = [self dateFromString:tf1.text];
```

```

NSDate *endDate = [self dateFromString:tf2.text];
NSDate *earlyDate = [beginDate earlierDate:endDate];
//判断开始时间和结束时间中的日期是否相等
if ([earlyDate isEqualToDate:endDate] && ![earlyDate isEqualToDate:
beginDate]) {
    [self alterMessage:@"开始时间不得晚于结束时间"];
    return;
}
NSDateComponents *beginComponets = [[NSCalendar autoupdatingCurrentCalendar]
components:NSWeekdayCalendarUnit fromDate:beginDate]; //创建日期组件
int beginWeekDay = [beginComponets weekday]; //获取日期组件是此周中的第几天
NSDateComponents *endComponets = [[NSCalendar autoupdatingCurrentCalendar]
components:NSWeekdayCalendarUnit fromDate:endDate]; //创建日期组件
int endWeekDay = [endComponets weekday]; //获取日期组件是此周中的第几天
//判断输入的日期是否是周末
if (beginWeekDay == 1 || beginWeekDay == 7 || endWeekDay == 1 || endWeekDay == 7) {
    [self alterMessage:@"结束或开始时间不得为周末"];
    return;
}
//实现计算
NSTimeInterval time = [endDate timeIntervalSinceDate:beginDate];
//获取时间

float oneWeekDay = 7 - beginWeekDay;
float allDay = time / (24 * 60 * 60); //获取所有的天数
float day = 0.0;
//判断所有的天数是否大于7天
if(allDay > oneWeekDay + 2){
    float otherDay = allDay - (oneWeekDay + 2);
    float ResidualDay = otherDay - ((int)otherDay / 7) * 2;
    day = ResidualDay + oneWeekDay;
}else{
    day = endWeekDay - beginWeekDay; //获取天数
}
dayLabel.text= [NSString stringWithFormat:@"%d", (int)day];
//设置标签中文本的内容
}

```

【代码解析】

本实例关键功能是工作日的计算。下面就是这个知识点的详细讲解。在本实例中，工作日的计算需要分为两种情况考虑。

1. 没有超过一周

在输入开始时间和结束时间后，首先需要判断开始时间和结束时间是否在同一周内，其代码如下：

```

if(allDay > oneWeekDay + 2){
    .....
}else{
    .....
}

```

如果输入的开始时间和结束时间在同一周内，就可以不需要考虑节假日进行计算，代码如下：

```

day = endWeekDay - beginWeekDay;

```

2. 超过一周

如果输入的开始时间和结束时间不在同一周内，就需要考虑到节假日，再进行计算，其代码如下：

```
float otherDay = allDay - (oneWeekDay + 2);
float ResidualDay = otherDay - ((int)otherDay / 7) * 2;
day = ResidualDay + oneWeekDay;
```

实例 93 短信发送

【实例描述】

在本实例实现的功能是一个短信发送的功能。当用户在文本框中输入中电话号码，并且在文本视图中输入内容后，单击“完成”按钮，就会自动打开系统自带的短信功能。这时，收件人以及短信内容都以及写好了，就是在文本框以及文本视图中的内容。单击“发送”按钮，短信就会被发送出去。运行效果如图 8.3 所示。

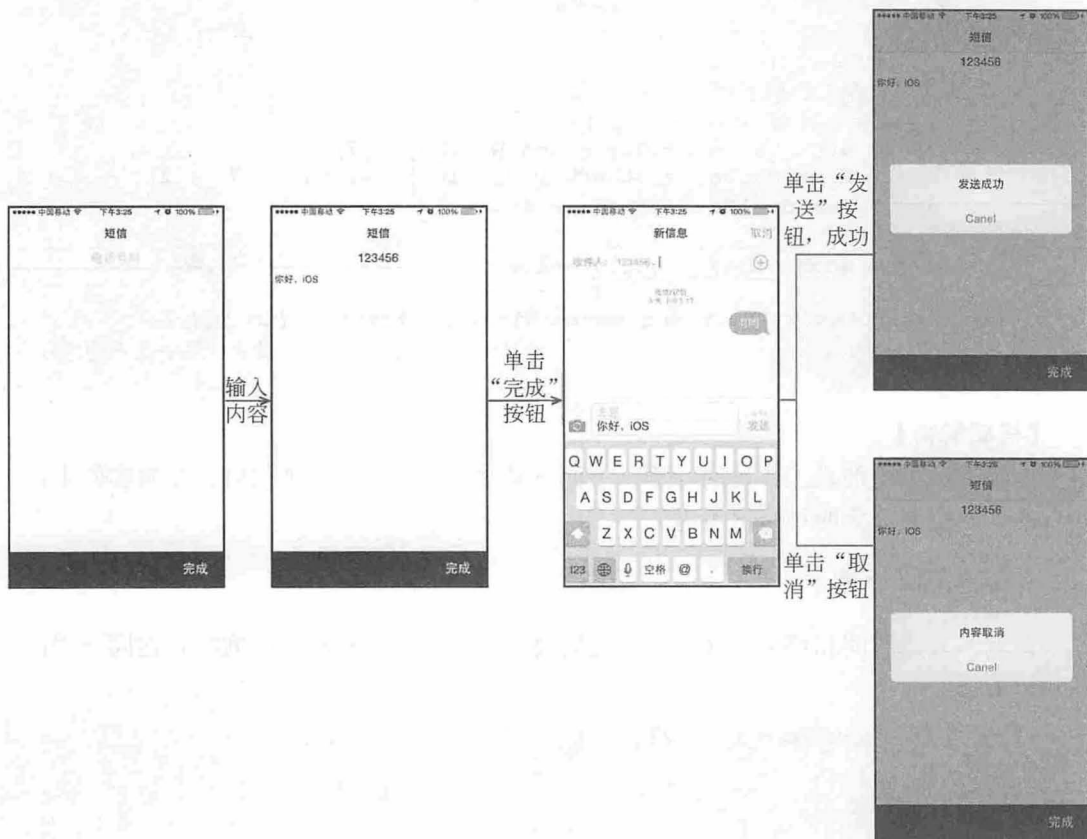


图 8.3 运行效果

【实现过程】

- (1) 创建一个项目，命名为“短信发送”。

- (2) 添加 MessageUI.framework 到创建的项目中。
- (3) 创建一个基于 NSObject 类的 ZCAddressBook 类。
- (4) 打开 ZCAddressBook.h 文件, 编写代码, 实现属性以及方法的声明。程序代码如下:

```
#import <Foundation/Foundation.h>
#import <MessageUI/MessageUI.h>
@interface ZCAddressBook : NSObject<MFMessageComposeViewControllerDelegate>
//属性
@property(nonatomic,assign) id target;
@property(nonatomic,copy) void(^MessageBlock)(int);
-(id)initWithTarget:(id)target MessageNameArray:(NSArray*)array Message:
(NSString*)str Block:(void (^)(int))a;
@end
```

- (5) 打开 ZCAddressBook.m 文件, 编写代码, 实现短信发送的功能。使用的方法如表 8-3 所示。

表 8-3 ZCAddressBook.m文件中方法总结

方 法	功 能
initWithTarget: MessageNameArray:Message: Block:	初始化
showViewMessageNameArray: Message:	打开内置的短信服务
messageComposeViewController: didFinishWithResult:	在用户单击发送或者取消时调用

其中, initWithTarget: MessageNameArray:Message: Block:方法实现一些初始化的设置。程序代码如下:

```
-(id)initWithTarget:(id)target MessageNameArray:(NSArray*)array Message:
(NSString*)str Block:(void (^)(int))a
{
    if (self=[super init]) {
        self.target=target;
        self.MessageBlock=a;
        [self showViewMessageNameArray:array Message:str]; //打开内置的短信服务
    }
    return self;
}
```

showViewMessageNameArray:Message:方法实现打开内置的短信服务, 并设置收件人以及短信内容。程序代码如下:

```
-(void)showViewMessageNameArray:(NSArray*)array Message:(NSString*)str{
    //判断当前设备是否可以发送信息
    if ([MFMessageComposeViewController canSendText]) {
        MFMessageComposeViewController *messageViewController = [[MFMessageComposeViewController alloc] init];
        //委托到本类
        messageViewController.messageComposeDelegate = self;
        //设置收件人
        messageViewController.recipients = array;
        //短信的内容
```

```

messageViewController.body =str;
//打开短信视图控制器
[self.target presentViewController:messageViewController animated:
YES completion:nil];
}
}

```

`messageComposeViewController:didFinishWithResult:`方法在用户单击发送或者取消时调用。程序代码如下：

```

- (void)messageComposeViewController:(MFMessageComposeViewController *)controller
didFinishWithResult:
(MessageComposeResult)result{
    if (result==0) {
        UIAlertView *alert=[[UIAlertView alloc]initWithTitle:@"内容取消"
        message:nil delegate:nil cancelButtonTitle:@"Canel"
        otherButtonTitles:nil]; //创建警告视图对象
        [alert show];
    }else if (result==1){
        UIAlertView *alert=[[UIAlertView alloc]initWithTitle:@"发送成功"
        message:nil delegate:nil cancelButtonTitle:@"Canel"
        otherButtonTitles:nil]; //创建警告视图对象
        [alert show];
    }else {
        UIAlertView *alert=[[UIAlertView alloc]initWithTitle:@"发送失败"
        message:nil delegate:nil cancelButtonTitle:@"Canel"
        otherButtonTitles:nil]; //创建警告视图对象
        [alert show];
    }
    self.MessageBlock(result);
    [controller dismissViewControllerAnimated:YES completion:nil];
}

```

(6) 打开 `ViewController.h` 文件，编写代码，实现头文件、插座变量以及动作的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "ZCAddressBook.h"
@interface ViewController : UIViewController{
    IBOutlet UITextField *tf; //声明关于文本框的插座变量
    IBOutlet UITextView *tv; //声明关于文本视图的插座变量
}
- (IBAction)button:(id)sender;
- (IBAction)hide:(id)sender;
@end

```

(7) 打开 `Main.storyboard` 文件，从视图库中拖动 `Navigation Controller` 导航控制器到画布中，并将控制器关联的根视图设置为 `View Controller` 视图控制器的视图，将 `Is Initial View Controller` 复选框选中。对 `View Controller` 视图控制器的设计界面进行设计，效果如图 8.4 所示。

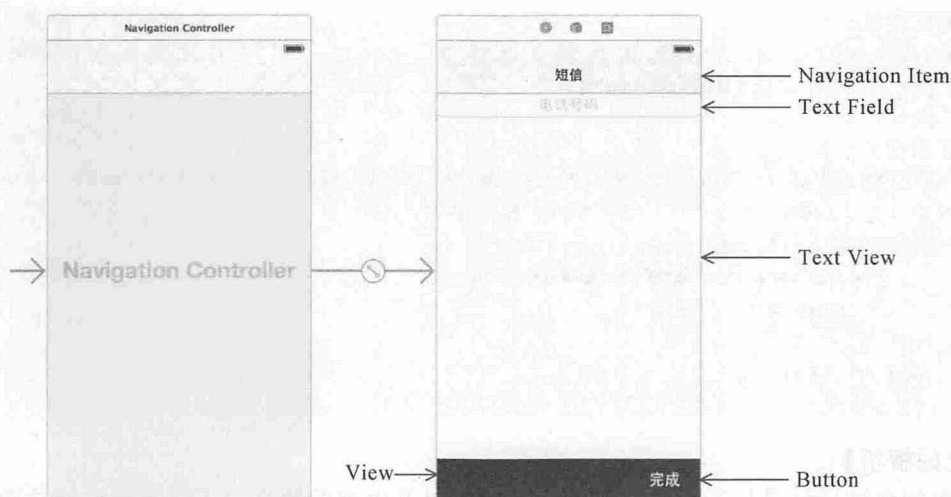


图 8.4 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 8-4 所示。

表 8-4 视图、控件设置

视图、控件	属性设置	其他
Navigation Item	Title: 短信	
Text Field	Alignment: 居中 Placeholder: 电话号码	与插座变量 tf 关联 与动作 hide 关联 关联 Did End On Exit
Text View	Text: (空) Font: System 19.0	
View	Background: 深灰色	
Button	Title: 完成 Font: System Bold 19.0 Text Color: 白色	与插座变量 dayLabel 关联

(8) 打开 ViewController.m 文件，编写代码，此代码实现的功能是信息的发送。

```
//单击“完成”按钮
- (IBAction)button:(id)sender {
    //判断 tf 和 tv 的长度是否为 0
    if(tf.text.length==0||tv.text.length==0){
        UIAlertView *a=[[UIAlertView alloc]initWithTitle:@"提示" message:@"
        请输入内容" delegate:nil cancelButtonTitle:@"Cancel"
        otherButtonTitles: nil]; //创建警告视图
        [a show];
    }else{
        [[ZCAddressBook alloc]initWithTarget:self MessageNameArray:@[tf.text]
        Message:tv.text Block:^(int type) {
            NSLog(@"发送短信后的状态"); //输出
        }];
    }
}
```

```

//隐藏键盘
- (IBAction)hide:(id)sender {
    [tf resignFirstResponder];
}
//隐藏键盘
- (BOOL)textView:(UITextView *)textView shouldChangeTextInRange:(NSRange)
range replacementText:(NSString *)text{
    if ([text isEqualToString:@"\n"]) {
        [textView resignFirstResponder]; //关闭键盘
        return NO;
    }
    return YES;
}

```

【代码解析】

本实例关键功能是打开系统自带的短信功能以及内容的填充。下面依次讲解这两个知识点。

1. 打开系统自带的短信功能

MessageUI 框架中的 MFMessageComposeViewController 类提供了发送短信的接口，它可以打开系统自带的短信功能。在本实例中，就是使用了此类打开了系统自带的短信功能。代码如下：

```

MFMessageComposeViewController *messageViewController = [[MFMessageComposeViewController alloc] init];
.....
[self.target presentViewController:messageViewController animated:YES completion:nil];

```

2. 内容的填充

MFMessageComposeViewController 类提供了很多的属性对短信功能界面的内容进行填充。最主要的两个是对收件人的填充以及短信内容的填充。其中，对收件人的内容填充需要使用 recipients 属性，其语法形式如下：

```
@property(nonaatomic, copy) NSArray *recipients
```

在本实例中，就是使用了此属性将文本框中输入的内容用来填充短信界面的收件人内容。代码如下：

```
messageViewController.recipients = array;
```

短信内容的填充需要使用 body 属性实现，其语法形式如下：

```
@property(nonaatomic, copy) NSString * body ;
```

在本实例中，就是使用了此属性将文本视图中输入的内容用来填充短信界面的短信内容。代码如下：

```
messageViewController.body =str;
```

实例 94 日 历

【实例描述】

iOS 系统自带的日历是没有农历的。但是对于中国人来说，农历是很重要的。本实例就为 iOS 系统中自动日历的不足进行了补充，实现了一个具有农历功能的实例。用户可以轻拍导航栏中的当前日期，后弹出一个选择日期视图，用户可以在此进行对日期的选择，当单击“确定”按钮，界面中的日历就会到达对应日期的界面。当单击“今天”按钮，界面就会回到当前的日期。运行效果如图 8.5 所示。

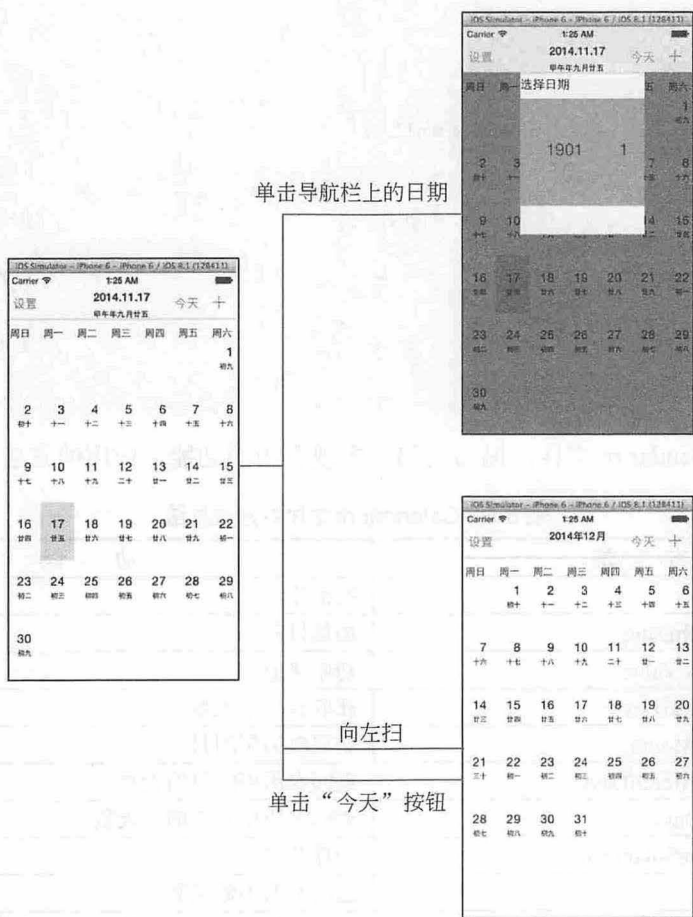


图 8.5 运行效果

【实现过程】

- (1) 创建一个项目，命名为“日历”。
- (2) 创建一个基于 NSObject 类的 Calendar 类。
- (3) 打开 Calendar.h 文件，编写代码，实现定义数据结构、对象、实例变量、方法的声明。程序代码如下：

```

#import <Foundation/Foundation.h>
//定义数据结构
struct SolarTerm
{
    __unsafe_unretained NSString *solarName;
    int solarDate;
};
@interface Calendar : NSObject{
    //对象
    NSArray *HeavenlyStems;
    NSArray *EarthlyBranches;
    .....
    NSString *zodiacLunar;
    NSString *solarTermTitle;
    //实例变量
    int year;
    int month;
    .....
    int lunarDay;
    struct SolarTerm solarTerm[2];
}
//方法
-(void)loadWithDate:(NSDate *)date;                //加载日期
-(void)InitializeValue;
-(int)LunarYearDays:(int)y;                          //获取农历年天数
.....
-(int)Weekday;                                       //返回一周的第几天
-(NSString *)Constellation;
@end

```

(4) 打开 Calendar.m 文件，编写代码，实现农历的功能，使用的方法如表 8-5 所示。

表 8-5 Calendar.m 文件中方法总结

方 法	功 能
init	初始化
loadWithDate:	加载日期
InitializeValue	初始化值
LunarYearDays:	获取农历年天数
DoubleMonth:	获取农历年闰月
DoubleMonthDays:	返回农历年闰月的天数
MonthDays::	返回农历年月份的总天数
ComputeSolarTerm	计算节气
Tail:	返回 x 的小数尾数
Term:::	返回 y 年第 n 个节气的日差天数
AntiDayDifference::	返回阳历 y 年日差天数为 x 时所对应的月日数
EquivalentStandardDay:::	获取等效的标准天数
DayDifference:::	返回阳历 y 年 m 月 d 日的日差天数
IfGregorian:::	判断 y 年 m 月 d 日是 Gregorian 历还是 Julian 历
MonthLunar	返回农历
DayLunar	返回农历日
ZodiacLunar	返回年生肖
YearHeavenlyStem	返回年天干

续表

方 法	功 能
MonthHeavenlyStem	返回月天干
DayHeavenlyStem	返回日天干
YearEarthlyBranch	返回年地支
MonthEarthlyBranch	返回月地支
DayEarthlyBranch	返回日地支
SolarTermTitle	返回节气
IsLeap	返回是不是农历闰年
GregorianYear	返回阳历年
GregorianMonth	返回阳历月
GregorianDay	返回阳历天
Weekday	返回一周的第几天
Constellation	返回星座

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，loadWithDate:方法实现数据的添加。程序代码如下：

```

-(void)loadWithDate:(NSDate *)adate
{
    //判断 adate 是否为空
    if (adate == nil)
        [self loadWithDate:[NSDate date]]; //加载日期
    else
    {
        //创建数组
        HeavenlyStems = [NSArray arrayWithObjects:@"甲",@"乙",@"丙",@"丁",
            @"戊",@"己",@"庚",@"辛",@"壬",@"癸",nil];
        EarthlyBranches = [NSArray arrayWithObjects:@"子",@"丑",@"寅",
            @"卯",@"辰",@"巳",@"午",@"未",@"申",@"酉",@"戌",@"亥",nil];
        .....
        arrayDay = [NSArray arrayWithObjects:@"初一", @"初二", @"初三",
            @"初四", @"初五", @"初六", @"初七", @"初八", @"初九", @"初十", @"十一",
            @"十二", @"十三", @"十四", @"十五", @"十六", @"十七", @"十八", @"十九",
            @"二十", @"廿一", @"廿二", @"廿三", @"廿四", @"廿五", @"廿六", @"廿七",
            @"廿八", @"廿九", @"三十", @"三十一", nil];
        //设置格式
        NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setDateStyle:NSDateFormatterMediumStyle];
        //设置日期的风格
        [dateFormatter setTimeStyle:NSDateFormatterNoStyle]; //设置时间的风格
        [dateFormatter setDateFormat:@"%yyyy"]; //设置日期的格式
        year = [[dateFormatter stringFromDate:adate] intValue];
        [dateFormatter setDateFormat:@"%MM"]; //设置日期的格式
        month = [[dateFormatter stringFromDate:adate] intValue];
        [dateFormatter setDateFormat:@"%dd"]; //设置日期的格式
        day = [[dateFormatter stringFromDate:adate] intValue];
        thisdate = adate;
    }
}

```

InitializeValue 方法实现对日历中的数据进行初始化的功能。程序代码如下：


```

-(void)InitializeValue
{
    NSString *start = @"1900-01-31";
    //设置格式
    NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
    [dateFormatter setDateFormat:@"%yyyy-MM-dd"]; //设置日期的格式
    NSString *end = [dateFormatter stringFromDate:thisdate];
    NSDateFormatter *f = [[NSDateFormatter alloc] init];
    [f setDateFormat:@"%yyyy-MM-dd"]; //设置日期的格式
    NSDate *startDate = [f dateFromString:start];
    NSDate *endDate = [f dateFromString:end]; //设置日期的格式
    NSCalendar *gregorianCalendar = [[NSCalendar alloc] initWithCalendar
    Identifier:NSGregorianCalendar];
    NSDateComponents *components = [gregorianCalendar components:NSDay
    CalendarUnit fromDate:startDate toDate:endDate options:0]; //创建日期组件
    int dayCyclical=(int)(([components day] + 30)/(86400/(3600*24)))+10;
    int sumdays = (int)[components day];
    int tempdays = 0;
    //计算农历年
    for (lunarYear = 1900; lunarYear < 2100 && sumdays > 0; lunarYear++){
        tempdays = [self LunarYearDays:lunarYear];
        sumdays -= tempdays;
    }
    //判断 sumdays 是否小于 0
    if (sumdays < 0){
        sumdays += tempdays;
        lunarYear--;
    }
    //计算闰月
    doubleMonth = [self DoubleMonth:lunarYear]; //获取农历年闰月
    isLeap = false;
    //计算农历月
    for (lunarMonth = 1; lunarMonth < 13 && sumdays > 0; lunarMonth++){
        //闰月
        if (doubleMonth > 0 && lunarMonth == (doubleMonth + 1) && isLeap ==
        false){
            --lunarMonth;
            isLeap = true;
            tempdays = [self DoubleMonthDays:lunarYear]; //返回农历年闰月的天数
        }else{
            tempdays = [self MonthDays:lunarYear:lunarMonth];
        }
        //取消闰月
        if (isLeap == true && lunarMonth == (doubleMonth + 1)){
            isLeap = false;
        }
        sumdays -= tempdays;
    }
    //计算农历日
    if (sumdays == 0 && doubleMonth > 0 && lunarMonth == doubleMonth + 1){
        //判断 isLeap 是否为真
        if (isLeap){
            isLeap = false;
        }
        else{
            isLeap = true;
            --lunarMonth;
        }
    }
}

```

```

    }
    //判断 sumdays 是否小于 0
    if (sumdays < 0){
        sumdays += tempdays;
        --lunarMonth;
    }
    lunarDay = sumdays + 1;
    //计算节气
    [self ComputeSolarTerm];
    solarTermTitle = @"";
    //循环
    for (int i=0; i<2; i++){
        NSDateFormatter *currentFormatter = [[NSDateFormatter alloc] init];
        [currentFormatter setDateFormat:@"%yyyyMMdd"]; //设置日期的格式
        if (solarTerm[i].solarDate == [[currentFormatter stringFromDate:
            thisdate] intValue])
            solarTermTitle = solarTerm[i].solarName;
    }
    monthLunar = (NSString *)[arrayMonth objectAtIndex:(lunarMonth - 1)];
    //每月农历
    dayLunar = (NSString *)[arrayDay objectAtIndex:(lunarDay - 1)];
    zodiacLunar = (NSString *)[LunarZodiac objectAtIndex:((lunarYear - 4)
    % 60 % 12)]; //生肖
    yearHeavenlyStem = (NSString *)[HeavenlyStems objectAtIndex:
    ((lunarYear - 4) % 60 % 10)];
    //判断
    if (((year-1900)*12+month+13)%10) == 0)
        monthHeavenlyStem = (NSString *)[HeavenlyStems objectAtIndex:9];
    else
        monthHeavenlyStem = (NSString *)[HeavenlyStems objectAtIndex:
        (((year-1900)*12+month+13)%10-1)];
    dayHeavenlyStem = (NSString *)[HeavenlyStems objectAtIndex:(dayCyclical%10)];
    yearEarthlyBranch = (NSString *)[EarthlyBranches objectAtIndex:
    ((lunarYear - 4) % 60 % 12)];
    //判断
    if (((year-1900)*12+month+13)%12) == 0)
        monthEarthlyBranch = (NSString *)[EarthlyBranches objectAtIndex:11];
    else
        monthEarthlyBranch = (NSString *)[EarthlyBranches objectAtIndex:
        (((year-1900)*12+month+13)%12-1)];
    dayEarthlyBranch = (NSString *)[EarthlyBranches objectAtIndex:
    (dayCyclical%12)];
}

```

ComputeSolarTerm 方法实现节气的计算。程序代码如下：

```

-(void)ComputeSolarTerm
{
    //遍历
    for (int n = month * 2 - 1; n <= month * 2; n++)
    {
        double Termdays = [self Term:year:n:YES];
        double mdays = [self AntiDayDifference:year:floor(Termdays)];
        int hour = (int)floor((double)[self Tail:Termdays] * 24); //小时
        int minute = (int)floor((double)([self Tail:Termdays] * 24 - hour)
        * 60); //分钟
        int tMonth = (int)ceil((double)n / 2);
        int tday = (int)mdays % 100;
    }
}

```

```

//判断
if (n >= 3)
    solarTerm[n - month * 2 + 1].solarName = [SolarTerms objectAtIndex:(n - 3)];
else
    solarTerm[n - month * 2 + 1].solarName = [SolarTerms objectAtIndex:(n + 21)];
//设置值
NSDateComponents *components = [[NSDateComponents alloc] init];
[components setYear:year]; //设置年
[components setMonth:tMonth]; //设置月
[components setDay:tday]; //设置天
[components setHour:hour]; //设置小时
[components setMinute:minute]; //设置分钟
NSCalendar *gregorian = [[NSCalendar alloc] initWithCalendarIdentifier:
    NSGregorianCalendar];
NSDate *ldate = [gregorian dateFromComponents:components];
NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
[dateFormatter setDateFormat:@"yyyyMMdd"]; //设置日期格式
solarTerm[n - month * 2 + 1].solarDate = [[dateFormatter stringFromDate:
    ldate] intValue];
}
}

```

Term:::方法实现获取 y 年第 n 个节气的日差天数。程序代码如下:

```

-(double)Term:(int)y :(int)n :(bool)pd
{
    //儒略日
    double juD = y * (365.2423112 - 6.4e-14 * (y - 100) * (y - 100) - 3.047e-8
    * (y - 100)) + 15.218427 * n + 1721050.71301;
    //角度
    double tht = 3e-4 * y - 0.372781384 - 0.2617913325 * n;
    //年差实均数
    double yrD = (1.945 * sin(tht) - 0.01206 * sin(2 * tht)) * (1.048994 -
    2.583e-5 * y);
    //朔差实均数
    double shuoD = -18e-4 * sin(2.313908653 * y - 0.439822951 - 3.0443 * n);
    double vs = (pd) ? (juD + yrD + shuoD - [self EquivalentStandardDay:y:1:0]
    - 1721425) : (juD - [self EquivalentStandardDay:y:1:0] - 1721425);
    return vs;
}

```

(5) 创建一个基于 NSDate 类的分类 Calendar。

(6) 打开 NSDate+Calendar.h 文件, 编写代码, 实现头文件以及方法的声明。程序代码如下:

```

#import <Foundation/Foundation.h>
#import "Calendar.h"
@interface NSDate (Calendar)
- (Calendar *)chineseCalendarDate;
@end

```

(7) 打开 NSDate+Calendar.m 文件, 编写代码, 实现中国日历的加载即农历的加载, 程序代码如下:

```

- (Calendar *)chineseCalendarDate
{
    Calendar *lunarCalendar = [[Calendar alloc] init];
    [lunarCalendar loadWithDate:self]; //加载日期
}

```

```
[lunarCalendar InitializeValue];
return lunarCalendar;
}
```

(8) 创建一个基于 NSObject 类的 JBCalendar 类。

(9) 打开 JBCalendar.h 文件, 编写代码, 实现属性以及方法的声明。程序代码如下:

```
#import <Foundation/Foundation.h>
@interface JBCalendar : NSObject
//属性
@property (nonatomic, assign) NSInteger year;
@property (nonatomic, assign) NSInteger month;
@property (nonatomic, assign) NSInteger day;
- (NSDate *)nsDate;
@end
```

(10) 打开 JBCalendar.m 文件, 编写代码, 对 nsDate 方法进行实现, 即实现使用 NSDateComponents 对象创建 NSDate 对象, 并返回。程序代码如下:

```
- (NSDate *)nsDate
{
    NSDateComponents *components = [[NSDateComponents alloc] init];
    components.year = self.year;
    components.month = self.month;           //获取月
    components.day = self.day;                //获取天
    return [[NSCalendar currentCalendar] dateFromComponents:components];
}
```

(11) 创建一个基于 NSObject 类的 Datetime 类。

(12) 打开 Datetime.h 文件, 编写代码, 实现方法的声明。程序代码如下:

```
#import <Foundation/Foundation.h>
@interface Datetime : NSObject
+ (NSArray *)GetAllYearArray;
.....
+ (NSString *)GetMinute;
+ (NSString *)GetSecond;
@end
```

(13) 打开 Datetime.m 文件, 编写代码, 实现获取日期时间的功能。使用的方法如表 8-6 所示。

表 8-6 Datetime.m文件中方法总结

方 法	功 能
GetAllYearArray	获取所有年列表
GetAllMonthArray	获取所有月列表
getDateTime	获取以 YYYY.MM.dd 格式输出年月日
GetDateTime	获取以 YYYY 年 MM 月 dd 日格式输出年月日
GetLunarDateTime	获取以 YYYY 年 MMdd 格式输出此时的农历年月日
GetDayArrayByYear:andMonth:	获取指定年份指定月份的星期排列表
GetLunarDayArrayByYear:andMonth:	获取指定年份指定月份的星期排列表 (农历)
GetLunarDayByYear:andMonth:andDay:	获取某年某月某日的对应农历日
GetTheWeekOfDayByYera:andByMonth:	计算 year 年 month 月第一天是星期几, 周日则为 0
GetNumberOfDayByYera:andByMonth:	判断 year 年 month 月有多少天
GetYear	获取年

续表

方 法	功 能
GetMonth	获取月
GetDay	获取天
GetHour	获取小时
GetMinute	获取分钟
GetSecond	获取秒

GetAllYearArray 方法获取从 1901 年到 2049 年所有的年份。程序代码如下：

```
+(NSArray *)GetAllYearArray{
    NSMutableArray * monthArray = [[NSMutableArray alloc] init];
    //循环，添加对象
    for (int i = 1901; i<2050; i++) {
        NSString * days = [[NSString alloc] initWithFormat:@"%d",i];
        [monthArray addObject:days]; //添加对象
    }
    return monthArray;
}
```

GetLunarDateTime 方法获取以 YYYY 年 MMdd 格式输出此时的农历年月日。程序代码如下：

```
+(NSString*)GetLunarDateTime{
    JBCalendar* date = [[JBCalendar alloc] init];
    date.year = [[self GetYear] intValue],date.month = [[self GetMonth]
    intValue],date.day = [[self GetDay] intValue];
    Calendar *lunarCalendar = [[date nsDate] chineseCalendarDate];
    NSString * lunar = [[NSString alloc] initWithFormat:@"%d年%d月%d日",
    lunarCalendar.YearHeavenlyStem,lunarCalendar.YearEarthlyBranch,
    lunarCalendar.MonthLunar,lunarCalendar.DayLunar]; //创建字符串对象
    return lunar; //返回字符串对象
}
```

GetDayArrayByYear::方法实现获取指定年份、月份的星期排列表。程序代码如下：

```
+(NSArray *)GetDayArrayByYear:(int) year andMonth:(int) month{
    NSMutableArray * dayArray = [[NSMutableArray alloc] init];
    //遍历
    for (int i = 0; i< 42; i++) {
        //判断 i 是否小于指定年份、月份的星期排列表
        if (i < [self GetTheWeekOfDayByYera:year andByMonth:month]-1) {
            [dayArray addObject:@" "]; //添加空对象
        }else if ((i>[self GetTheWeekOfDayByYera:year andByMonth:month]-1)&&
        (i<[self GetTheWeekOfDayByYera:year andByMonth:month]+[self GetNumber
        OfDayByYera:year andByMonth:month])){
            NSString * days;
            if((i - [self GetTheWeekOfDayByYera:year andByMonth:month] +1)< 10)
                days = [NSString stringWithFormat:@"%d",i-[self GetTheWeek
                OfDayByYera:year andByMonth:month]+1]; //创建字符串对象
            else
                days = [NSString stringWithFormat:@"%d",i-[self GetTheWeek
                OfDayByYera:year andByMonth:month]+1];
            [dayArray addObject:days]; //添加对象
        }else {
            [dayArray addObject:@" "]; //添加对象
        }
    }
}
```



```

    }
}
return dayArray;
}

```

GetLunarDayByYear:andMonth:andDay:方法实现获取某年某月某日的对应农历日。程序代码如下:

```

+ (NSString *)GetLunarDayByYear: (int) year andMonth: (int) month andDay: (int) day{
    JBCalendar* date = [[JBCalendar alloc] init]; //创建 JBCalendar 对象
    date.year = year, date.month = month, date.day = day;
    Calendar *lunarCalendar = [[date nsDate] chineseCalendarDate];
    NSString * lunarday = [[NSString alloc] initWithString:lunarCalendar.
    DayLunar]; //创建字符串对象
    return lunarday;
}

```

GetTheWeekOfDayByYera:andByMonth:方法计算 year 年 month 月第一天是星期几, 周日则为 0。程序代码如下:

```

+ (int) GetTheWeekOfDayByYera: (int) year andByMonth: (int) month{
    int numWeek = ((year-1)+ (year-1)/4- (year-1)/100+ (year-1)/400+1)%7;
    //numWeek 为 years 年的第一天是星期几
    int ar[12] = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334};
    //numdays 为 month 月 years 年的第一天是这一年的第几天
    int numdays = (((year/4==0&&year/100!=0) || (year/400==0)) && (month>2)) ?
    (ar[month-1]+1): (ar[month-1]);
    int dayweek = (numdays%7 + numWeek)%7; //month 月第一天是星期几, 周日则为 0
    return dayweek;
}

```

(14) 打开 ViewController.h 文件, 编写代码, 实现头文件、遵守协议、对象、实例变量以及属性的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "Datetime.h"
@interface ViewController : UIViewController<UIPickerViewDelegate,
UIPickerViewDataSource>{
    //对象
    NSArray * dayArray;
    NSArray * lunarDayArray;
    UIView * timePickerView;
    //实例变量
    int strMonth;
    int strYear;
    bool timePacker;
}
//属性
@property(n nonatomic, retain) UIPickerView *pickerView;
@property(retain, nonatomic) UIView *titleLabel;
@end

```

(15) 打开 Main.storyboard 文件, 从视图库中拖动 Navigation Controller 导航控制器到画布中, 并将控制器关联的根视图设置为 View Controller 视图控制器的视图, 将 Is Initial View Controller 复选框选中。

(16) 打开 ViewController.m 文件, 编写代码, 实现将带有农历的日历显示在界面上。

使用的方法如表 8-7 所示。

表 8-7 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
AddNavigationBarToRootView	添加导航栏
AddTimeLableToNavigationBar	为导航栏添加中间的时间标题
CalendarTitleLabel	制作显示阳历的标签
LunarCalendarTitleLabel	制作显示阴历的标签
titleButtenAction:	时间标题点击事件
AddLeftButtenToNavigationBar	为导航栏添加左边的设置按钮
LeftButtenAction	单击左边设置按钮后的动作
AddRightButtenArrayToNavigationBar	为导航栏添加右边的返回今天按钮和添加事件的按钮
backTodayAction	单击返回今天按钮后的动作
eventAddAction	单击添加事件按钮后的动作
AddTimePickerToCalendarWatch	添加时间选择器
ensureButtenAction:	时间选择器的确定和返回按钮
backButtenAction:	单击时间选择器中返回按钮后的动作
numberOfComponentsInPickerView:	时间选择器的栏数
pickerView:numberOfRowsInComponent:	为时间选择器添加数据源
pickerView:titleForRow:	将数据源显示在视图上
pickerView:widthForComponent:	两栏宽度
reMoveTimePickerToCalendarWatch	移除时间选择器
AddWeekLableToCalendarWatch	向日历中添加星期标号（周日到周六）
AddDaybuttenToCalendarWatch	向日历中添加指定月份的日历按钮
reloadDaybuttenToCalendarWatch	重新加载指定月份的日历按钮
buttenTouchUpInsideAction:	添加指定月份的日历按钮后的动作
AddHandleSwipe	添加滑动手势
leftHandleSwipe:	左滑事件
rightHandleSwipe:	右滑事件
reloadDateForCalendarWatch	重新加载数据

其中，viewDidLoad 方法在视图加载后调用，实现初始化的功能。程序代码如下：

```

-(void)viewDidLoad{
    //获取转换数据
    [super viewDidLoad];
    strYear = [[Datetime GetYear] intValue];
    strMonth = [[Datetime GetMonth] intValue];           //转换为整型
    dayArray = [Datetime GetDayArrayByYear:strYear andMonth:strMonth];
    lunarDayArray = [Datetime GetLunarDayArrayByYear:strYear andMonth:
    strMonth];
    timePacker = YES;
    //添加内容
    [self AddNavigationBarToRootView];           //添加导航栏
    [self AddWeekLableToCalendarWatch];
    [self AddDaybuttenToCalendarWatch];
    [self AddHandleSwipe];           //添加滑动手势
}

```

CalendarTitleLabel 方法实现使用标签显示阳历内容的功能。程序代码如下:

```
-(UILabel *)CalendarTitleLabel{
    //创建并设置标签对象
    UILabel* calendarTitleLabel = [[UILabel alloc] initWithFrame:CGRectMake(
    0, 0 ,80, 24)];
    calendarTitleLabel.backgroundColor = [UIColor clearColor];
    //设置 Label 背景透明度
    calendarTitleLabel.font = [UIFont boldSystemFontOfSize:20];
    //设置文本字体与大小
    calendarTitleLabel.textColor = [UIColor blackColor];
    //判断
    if ((strYear == [[Datetime GetYear] intValue]) && (strMonth == [[Datetime
    GetMonth]
    intValue])){
        calendarTitleLabel.text = [Datetime getDateTime]; //设置文本内容
    }else {
        //判断 strMonth 是否大于 10
        if (strMonth < 10) {
            calendarTitleLabel.text = [NSString stringWithFormat:@"%d 年 %d
            月",strYear,strMonth];
        }else{
            calendarTitleLabel.text = [NSString stringWithFormat:@"%d 年%d
            月",strYear,strMonth];
        }
    }
    calendarTitleLabel.hidden = NO;
    calendarTitleLabel.tag = 2001; //设置 tag 值
    calendarTitleLabel.adjustsFontSizeToFitWidth = YES;
    return calendarTitleLabel;
}
```

LunarCalendarTitleLabel 方法实现使用标签显示农历内容的功能。程序代码如下:

```
-(UILabel *)LunarCalendarTitleLabel{
    //创建并设置标签对象
    UILabel* lunarTitleLabel = [[UILabel alloc] initWithFrame:CGRectMake(0,
    24 ,80, 20)];
    lunarTitleLabel.backgroundColor = [UIColor clearColor];
    //设置 Label 背景透明
    lunarTitleLabel.font = [UIFont boldSystemFontOfSize:20];
    //设置文本字体与大小
    lunarTitleLabel.textColor = [UIColor blackColor];
    if ((strYear == [[Datetime GetYear] intValue]) && (strMonth == [[Datetime
    GetMonth] intValue])){
        lunarTitleLabel.text = [Datetime GetLunarDateTime];
    }else{
        lunarTitleLabel.hidden = NO; //不隐藏标题
    }
    lunarTitleLabel.tag = 2002; //设置 tag 值
    lunarTitleLabel.adjustsFontSizeToFitWidth = YES;
    return lunarTitleLabel;
}
```

AddTimePickerToCalendarWatch 方法实现为日历中添加日期选择器。程序代码如下:

```
-(void)AddTimePickerToCalendarWatch{
    //创建并设置空白视图对象
```

```

timePickerView = [[UIView alloc] initWithFrame:CGRectMake(0, -30, 320, 600)];
timePickerView.hidden = timePacker;
timePickerView.backgroundColor = [UIColor colorWithRed:0 green:0 blue:0
alpha:0.5]; //设置背景颜色
timePickerView.tag = 1000; //设置 tag 值
UIView *timePicker = [[UIView alloc] init];
timePicker.frame = CGRectMake(80, 80, 173, 240);
timePicker.backgroundColor = [UIColor greenColor]; //设置背景颜色
//创建并设置标签对象
UILabel * label = [[UILabel alloc] initWithFrame:CGRectMake(0, 10, 173, 40)];
label.text = @"选择日期";
label.backgroundColor = [UIColor whiteColor];
label.textColor = [UIColor blackColor]; //设置文本颜色
//创建并设置自定义选择器
pickerView = [[UIPickerView alloc] initWithFrame:CGRectMake(0, 40, 173, 100)];
pickerView.delegate=self; //设置委托
pickerView.showsSelectionIndicator=YES;
//创建并设置按钮对象
UIButton * ensureButten = [[UIButton alloc] initWithFrame:CGRectMake(0,
200, 80, 40)];
[ensureButten addTarget:self action:@selector(ensureButtenAction:)
forControlEvents:UIControlEventTouchUpInside]; //添加动作
[ensureButten setTitle:@"确定" forState:UIControlStateNormal];
ensureButten.titleLabel.textColor = [UIColor blackColor];
[ensureButten setBackgroundColor:[UIColor whiteColor]];
UIButton * backButten = [[UIButton alloc] initWithFrame:CGRectMake(80,
200, 93, 40)];
[backButten setTitle:@"返回" forState:UIControlStateNormal]; //设置标题
[backButten addTarget:self action:@selector(backButtenAction:)
forControlEvents:UIControlEventTouchUpInside];
backButten.titleLabel.textColor = [UIColor blackColor];
[backButten setBackgroundColor:[UIColor whiteColor]]; //设置背景颜色
//添加视图和控件
[timePicker addSubview:label];
[timePicker addSubview:ensureButten];
[timePicker addSubview:backButten];
[timePicker addSubview:pickerView];
[timePickerView addSubview:timePicker];
[self.view addSubview:timePickerView];
}

```

AddDaybuttonToCalendarWatch 方法实现为日历中添加指定月份的日历按钮。程序代码如下：

```

-(void)AddDaybuttonToCalendarWatch{
    for (int i = 0; i < 42; i++) {
        //创建并设置按钮对象
        UIButton * button = [[UIButton alloc] init];
        button.frame = CGRectMake((i%7)*47, 60+20+(i/7)*80, 47, 80); //设置框架
        if ([[Datetime GetDay] intValue]== [dayArray[i] intValue])&&
            (strMonth == [[Datetime GetMonth] intValue])&& (strYear == [[Datetime
GetYear] intValue]) {
            button.backgroundColor = [UIColor yellowColor]; //设置背景颜色
        }
        [button setTag:i+301];
        [button addTarget:self action:@selector(buttonTouchUpInsideAction:)
forControlEvents:UIControlEventTouchUpInside]; //添加动作
        button.showsTouchWhenHighlighted = YES;
    }
}

```

```

//创建并设置标签对象
UILabel* label = [[UILabel alloc] init];
label.text = [NSString stringWithString:dayArray[i]];
label.font=[UIFont fontWithName:@"Verdadana" size:30];
label.textColor = [UIColor blackColor];           //设置文本颜色
label.backgroundColor = [UIColor clearColor];
label.textAlignment=NSTextAlignmentCenter;
label.frame = CGRectMake(0, 0, 47, 60);           //设置框架
UILabel* lurLabel = [[UILabel alloc] init];
lurLabel.text = [NSString stringWithString:lunarDayArray[i]];
lurLabel.font=[UIFont boldSystemFontOfSize:10];
lurLabel.textColor = [UIColor blackColor];         //设置文本颜色
lurLabel.backgroundColor = [UIColor clearColor];
lurLabel.textAlignment=NSTextAlignmentCenter;     //设置对齐方式
lurLabel.frame = CGRectMake(0, 40, 47, 20);
//添加创建的对象
[button addSubview:label];
[button addSubview:lurLabel];
[self.view addSubview:button];
}
}

```

buttonTouchUpInsideAction:方法实现单击指定月份的日历按钮后实现的响应。程序代码如下:

```

- (void)buttonTouchUpInsideAction:(id) sender{
    NSInteger t = [sender tag]-301;
    dayArray = nil, lunarDayArray = nil;
    dayArray = [Datetime GetDayArrayByYear:strYear andMonth:strMonth];
    lunarDayArray = [Datetime GetLunarDayArrayByYear:strYear andMonth:strMonth];
    NSLog(@"%d 年%d 月%d 日", strYear, strMonth, dayArray[t]); //输出
    NSLog(@"%@", lunarDayArray[t]);
}

```

leftHandleSwipe:方法实现了向左扫的手势功能。程序代码如下:

```

- (void)leftHandleSwipe:(UISwipeGestureRecognizer *)gestureRecognizer {
    strMonth = strMonth+1;
    //判断 strMonth 是否等于 13
    if(strMonth == 13){
        strYear++;
        strMonth = 1;
    }
    [self reloadDataForCalendarWatch];
}

```

【代码解析】

本实例关键功能是农历的显示以及农历和日历的对应关系。下面依次讲解这两个知识点。

1. 农历的显示

在本实例农历的显示是使用标签进行实现的,代码如下:

```

UILabel* lunarTitleLabel = [[UILabel alloc] initWithFrame:CGRectMake(0,
24, 80, 20)];
.....
if ((strYear == [[Datetime GetYear] intValue]) && (strMonth == [[Datetime

```



```
GetMonth] intValue]))){
    lunarTitleLabel.text = [Datetime GetLunarDateTime];
}else{
    lunarTitleLabel.hidden = NO;//设置标题
}
.....
return lunarTitleLabel;
```

2. 农历和日历的对应关系

在本实例中农历和日历的对应关系是通过 GetLunarDayByYear:andMonth:andDay:方法实现的。代码如下：

```
JBCalendar* date = [[JBCalendar alloc] init];
date.year = year,date.month = month,date.day = day;
Calendar *lunarCalendar = [[date nsDate] chineseCalendarDate];
NSString * lunarday = [[NSString alloc] initWithString:lunarCalendar.DayLunar];
return lunarday;
```

实例 95 添 加 录

【实例描述】

本实例实现的功能是一个添加录的功能。当用户单击界面的 Add New Contact 按钮后，弹出添加联系人界面，当联系人信息添加好以后，单击 Done 按钮，联系人的相关信息就会保存。单击“通讯录”按钮，就可以查看保存的信息。运行效果如图 8.6 所示。

添加完联系人后，回到主界面，单击“通讯录”按钮



图 8.6 运行效果

【实现过程】

- (1) 创建一个项目，命名为“添加录”。
- (2) 添加 AddressBookUI.framework 到创建的项目中。
- (3) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议以及动作的声明。

程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AddressBookUI/AddressBookUI.h>
@interface ViewController : UIViewController<ABPeoplePickerNavigation
ControllerDelegate,
ABNewPersonViewControllerDelegate>{
}
//动作
- (IBAction)add:(id) sender;           //添加
- (IBAction)show:(id) sender;        //显示
@end
```

(4) 打开 Main.storyboard 文件，从视图库中拖动 Navigation Controller 导航控制器到画布中，并将控制器关联的根视图设置为 View Controller 视图控制器的视图，将 Is Initial View Controller 复选框选中。对 View Controller 视图控制器的设计界面进行设计，效果如图 8.7 所示。

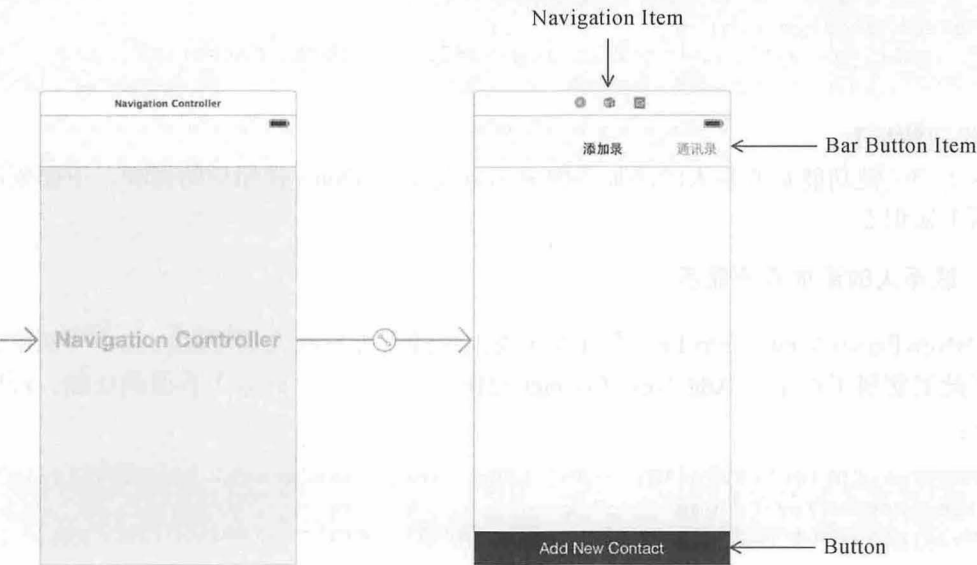


图 8.7 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 8-8 所示。

表 8-8 视图、控件设置

视图、控件	属 性 设 置	其 他
Navigation Item	Title: 添加录	
Bar Button Item	Title: 通讯录	与动作 show:关联
Button	Title: Add New Contact Font: System Bold 20.0 Text Color: 白色	与动作 add:关联

(5) 打开 ViewController.m 文件，编写代码，实现添加联系人的功能。程序代码如下：

```
//单击 Add New Contact 按钮
- (IBAction)add:(id) sender {
    ABNewPersonViewController *aa=[[ABNewPersonViewController alloc] init];
```

```

    aa.newPersonViewDelegate=self;                                //设置委托
    [self.navigationController pushViewController:aa animated:YES];
}
//关闭地址簿
-(void)peoplePickerNavigationControllerDidCancel:(ABPeoplePickerNavigationController *)peoplePicker{
    [self dismissViewControllerAnimated:YES completion:NO];
}
//单击“通讯录”按钮
- (IBAction)show:(id)sender {
    ABPeoplePickerNavigationController *people=[[ABPeoplePickerNavigationController alloc]init];
    people.peoplePickerDelegate=self;                            //设置委托
    [self presentViewController:people animated:YES completion:NO];
}
//实现数据保存
-(void)newPersonViewController:(ABNewPersonViewController *)newPersonView didCompleteWithNewPerson:(ABRecordRef)person{
    [self.navigationController popViewControllerAnimated:YES];
}

```

【代码解析】

本实例关键功能是联系人的添加界面显示以及单击 Done 按钮后的保存。下面依次讲解这两个知识点。

1. 联系人的添加界面显示

ABNewPersonViewController 类可以实现地址簿添加联系人的功能，在本实例中就是使用了此类实现了在单击 Add New Contact 按钮后，出现添加联系人界面的功能。程序代码如下：

```

ABNewPersonViewController *aa=[[ABNewPersonViewController alloc]init];
aa.newPersonViewDelegate=self;
[self.navigationController pushViewController:aa animated:YES];

```

2. 单击Done按钮后的保存

ABNewPersonViewControllerDelegate 协议中的 newPersonViewController:didCompleteWithNewPerson:方法实现在用户单击添加界面的保存或者取消按钮的响应。当用户单击保存按钮，当前的联系人信息会保存在地址簿中。其语法形式如下：

```

- (void)newPersonViewController:(ABNewPersonViewController *)newPersonViewController didCompleteWithNewPerson:(ABRecordRef)person;

```

其中，(ABNewPersonViewController *)newPersonViewController 表示添加联系人界面；didCompleteWithNewPerson:(ABRecordRef)person 表示当保存时，保存新创建联系人的记录，当取消时，为空。在本实例中就使用了 newPersonViewController:didCompleteWithNewPerson:方法实现了单击联系人的添加界面上的 Done 按钮后保存信息的功能。代码如下：

```

-(void)newPersonViewController:(ABNewPersonViewController *)newPersonView didCompleteWithNewPerson:(ABRecordRef)person{
    [self.navigationController popViewControllerAnimated:YES];
}

```

第9章 触摸和手势

在具有 iOS 系统的 iPhone 或者 iPad 中，为了给用户展示更多的屏幕，通常都不会显示键盘。用户对手机的操作统统都是通过触摸或者手势进行实现的。本章就为各位读者实现一些在 iOS 中常出现的手势实例。

实例 96 打地鼠

【实例描述】

打地鼠是一个富有趣味性的休闲游戏。本实例就为读者实现打地鼠的基本功能，当用户打到地鼠，就会实现对应的加分以及音乐的播放。运行效果如图 9.1 所示。



图 9.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“打地鼠”。
- (2) 添加 AudioToolbox.framework 到创建的项目中。
- (3) 添加图像 1.jpg、2.png 到创建项目的 Supporting Files 文件夹中。
- (4) 添加音频文件 3.mp3 到创建项目的 Supporting Files 文件夹中。
- (5) 打开 ViewController.h 文件，编写代码，实现头文件、插座变量以及实例变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AudioToolbox/AudioToolbox.h>
@interface ViewController : UIViewController{
    IBOutlet UIImageView *imageView;           //声明关于图像视图的插座变量
    IBOutlet UILabel *label;                  //声明关于标签的插座变量
    int i;
}
@end
```

- (6) 打开 main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，

效果如图 9.2 所示。



图 9.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 9-1 所示。

表 9-1 视图、控件设置

视图、控件	属 性 设 置	其 他
Label1	Text: 分数: Color: 白色 Font: System 19.0	
Label2	Text: 0 Color: 白色 Font: System 22.0	与动作 show:关联
Image View1	Image: 2.png	与动作 add:关联
Image View2	Image: 1.jpg	

(7) 打开 ViewController.m 文件，编写代码，实现打地鼠的功能。使用的方法如表 9-2 所示。

表 9-2 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
aa:	实现轻拍
show	显示地鼠的位置

需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewDidLoad 方法在视图加载后调用，实现对手势识别器对象以及定时器对象的创建。程序代码如下：

```
- (void)viewDidLoad{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    UITapGestureRecognizer *tap=[[UITapGestureRecognizer alloc] initWithTarget:
self action:@selector(aa:)]; //创建轻拍手势识别器对象
[self.view addGestureRecognizer:tap];
[NSTimer scheduledTimerWithTimeInterval:1 target:self selector:@selector
(show) userInfo:nil repeats:YES]; //创建时间定时器
}
```

aa:方法实现轻拍的功能，并判断轻拍的位置是否和地鼠的位置一致。程序代码如下：

```
-(void) aa: (UITapGestureRecognizer *)recognizer{
```



```

CGPoint aa=[recognizer locationInView:self.view];
if (imageView.frame.origin.x<=aa.x&&aa.x<=imageView.frame.origin.x+64&&
imageView.frame.origin.y<=aa.y&&aa.y<=imageView.frame.origin.y+66) {
    //播放系统声音
    SystemSoundID systemsoundID;
    NSString *file=[[NSBundle mainBundle]pathForResource:@"3" ofType:@"mp3"];
    AudioServicesCreateSystemSoundID((__bridge CFURLRef) [NSURL
        fileURLWithPath:file], &systemsoundID);           //创建系统声音
    AudioServicesPlaySystemSound(systemsoundID);           //播放系统声音
    //实现加分的功能
    i++;
    label.text=[NSString stringWithFormat:@"%i",i];         //设置显示文本
}
}

```

【代码解析】

本实例关键功能是打地鼠的实现以及打到地鼠加分的判断。下面依次讲解这两个知识点。

1. 打地鼠的实现

UITapGestureRecognizer 手势识别器的功能是实现轻拍的手势。在本实例中就使用了 UITapGestureRecognizer 实现了打地鼠的功能。代码如下：

```

UITapGestureRecognizer *tap=[ [UITapGestureRecognizer alloc] initWithTarget:
self action:@selector(aa:)];
[self.view addGestureRecognizer:tap];

```

2. 打到地鼠加分的判断

UITouch 的 locationInView:方法对视图中当前位置进行获取，其语法形式如下：

```

- (CGPoint)locationInView:(UIView *)view;

```

其中，(UIView *)view 表示视图对象。在本实例中就是使用了此方法对手势所在的当前位置进行获取，代码如下：

```

CGPoint aa=[recognizer locationInView:self.view];

```

其中，self.view 表示视图对象。使用获取的位置和图像视图即地鼠的位置进行判断。如果在图像视图的区域内，就表示打到地鼠了。代码如下：

```

if (imageView.frame.origin.x<=aa.x&&aa.x<=imageView.frame.origin.x+64&&
imageView.frame.origin.y<=aa.y&&aa.y<=imageView.frame.origin.y+66) {
    .....
    i++;
    label.text=[NSString stringWithFormat:@"%i",i];
}

```

实例 97 人鱼公主换发记

【实例描述】

换装、换发的游戏想必是很多小女生都很喜欢的。本实例就为使用换装、换发的游戏

基本原理来实现人鱼公主的换发。当用户将头发拖动到指定的位置后，停止触摸，人鱼公主的发型就会改变为相应的发型；当用户拖动的头发不在指定的位置时，头发会放回原来的位置。运行效果如图 9.3 所示。

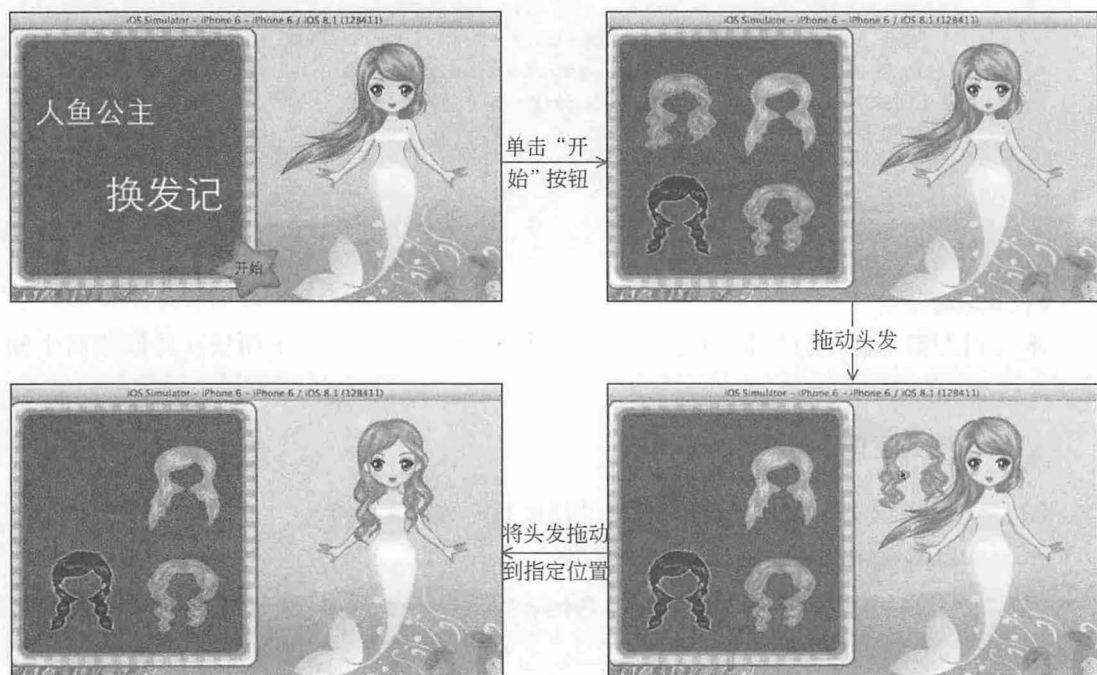


图 9.3 运行效果

【实现过程】

- (1) 创建一个项目，命名为“人鱼公主换发记”。
- (2) 添加图像 1.png~10.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIButton 类的 Button 类。
- (4) 打开 Button.h 文件，编写代码，实现协议、实例变量以及属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
//协议
@protocol ButtonDelegate <NSObject>
-(void)change; //对按钮的位置进行判断，从而实现换发的功能
-(void)back; //设置按钮的框架
@end
@interface Button : UIButton{
    CGPoint beginPoint; //实例变量
}
//属性
@property (nonatomic) BOOL dragEnable;
@property (nonatomic, assign) id <ButtonDelegate> delegate;
@end
```

- (5) 打开 Button.m 文件，编写代码，实现触摸功能。程序代码如下：

```

//开始触摸
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (!dragEnable) {
        return;
    }
    UITouch *touch = [touches anyObject];
    beginPoint = [touch locationInView:self];           //获取触摸点
}
//移动触摸
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
    if (!dragEnable) {
        return;
    }
    UITouch *touch = [touches anyObject];
    CGPoint nowPoint = [touch locationInView:self];    //获取触摸点
    //设置偏移量
    float offsetX = nowPoint.x - beginPoint.x;
    float offsetY = nowPoint.y - beginPoint.y;
    self.center = CGPointMake(self.center.x + offsetX, self.center.y +
                                offsetY);              //设置中心位置
}
//触摸结束
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
    //判断中心位置 x 的值是否大于 378 小于 450
    if (378 < self.center.x && self.center.x < 450) {
        self.hidden = YES;
        [self.delegate change];                       //调用 change 方法
    } else {
        [self.delegate back];                         //调用 back 方法
    }
}

```

(6) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 9.4 所示。



图 9.4 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 9-3 所示。

表 9-3 视图、控件设置

视图、控件	属 性 设 置	其 他
Label1	Text: 人鱼公主 Color: 白色 Font: System 35.0 Alignment: 居中	
Label2	Text: 换发记 Color: 白色 Font: System 46.0 Alignment: 居中	
Button	Title: 开始 Font: System 17.0 Text Color: 白色 Background: 黑色	
Image View	Image: 9.png	

(7) 创建一个基于 UIViewController 类的 AAViewController 类。

(8) 打开 AAViewController.h 文件，编写代码，实现头文件、遵守协议以及插座变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "Button.h"
@interface AAViewController : UIViewController<ButtonDelegate>{
    //插座变量
    IBOutlet Button *button1;
    IBOutlet Button *button2;
    IBOutlet Button *button3;
    IBOutlet Button *button4;
    IBOutlet UIImageView *backgroundImageView;
}
@end
```

(9) 打开 Main.storyboard 文件，从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 AAViewController。这时新增的 View Controller 视图控制器就变为了 AA View Controller 视图控制器。将 View Controller 中的按钮与此视图控制器进行关联。

(10) 对 AA View Controller 视图控制器的设计界面进行设计，效果如图 9.5 所示。



图 9.5 AA View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 9-4 所示。

表 9-4 视图、控件设置

视图、控件	属 性 设 置	其 他
Button1	Title: (空) Background: 1.png	Class: Button 与插座变量 button1 关联
Button2	Title: (空) Background: 2.png	Class: Button 与插座变量 button2 关联
Button3	Title: (空) Background: 3.png	Class: Button 与插座变量 button3 关联
Button4	Title: (空) Background: 4.png	Class: Button 与插座变量 button4 关联
Image View	Image: 9.png	与插座变量 backgroundImageView 关联

(11) 打开 AAViewController.m 文件, 编写代码, 实现人鱼公主换发的功能。程序代码如下:

```
//视图加载后调用, 实现初始化
- (void)viewDidLoad{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    button1.delegate=self; //设置委托
    button1.dragEnable=YES;
    .....
    button4.delegate=self;
    button4.dragEnable=YES;
}
//对按钮的位置进行判断, 从而实现换发的功能
- (void)change{
    //判断按钮的中心位置 x 的值是否大于 378 小于 450
    if (378<button1.center.x&&button1.center.x<450) {
        backgroundImageView.image=[UIImage imageNamed:@"5.png"]; //设置图像
        button2.hidden=NO;
        button3.hidden=NO;
        button4.hidden=NO;
        [self back];
    }else if (378<button2.center.x&&button2.center.x<450){
        //判断按钮的中心位置 x 的值是否大于 378 小于 450
        backgroundImageView.image=[UIImage imageNamed:@"6.png"]; //设置图像
        .....
        [self back];
    }else if (378<button3.center.x&&button3.center.x<450){
        //判断按钮的中心位置 x 的值是否大于 378 小于 450
        backgroundImageView.image=[UIImage imageNamed:@"7.png"];
        .....
        [self back];
    }else if (378<button4.center.x&&button4.center.x<450){
        //判断按钮的中心位置 x 的值是否大于 378 小于 450
        backgroundImageView.image=[UIImage imageNamed:@"8.png"];
        .....
        [self back];
    }
}
//设置按钮的框架
```



```

-(void)back{
    button1.frame=CGRectMake(40, 45, 89, 106);
    button2.frame=CGRectMake(156, 45, 89, 106);
    button3.frame=CGRectMake(40, 169, 89, 106);
    button4.frame=CGRectMake(156, 169, 89, 106);
}

```

【代码解析】

本实例关键功能是头发的拖动以及换发的功能。下面依次讲解这两个知识点。

1. 头发的拖动

在本实例中头发的拖动是使用触摸事件中的 `touchesMoved:withEvent:` 方法实现了拖动的移动功能。代码如下：

```

UITouch *touch = [touches anyObject];
CGPoint nowPoint = [touch locationInView:self];
float offsetX = nowPoint.x - beginPoint.x;
float offsetY = nowPoint.y - beginPoint.y;
self.center = CGPointMake(self.center.x + offsetX, self.center.y + offsetY);

```

2. 换发

当用户将头发拖动到指定的位置后，停止触摸，人鱼公主的发型就会改变；当用户拖动的头发不在指定的位置时，头发会放回原来的位置。它的实现是通过触摸事件中的 `touchesEnded:withEvent:` 方法实现的。代码如下：

```

if (378<self.center.x&&self.center.x<450) {
    self.hidden=YES;
    [self.delegate change];
}else{
    [self.delegate back];
}

```

实例 98 被挤扁的气球

【实例描述】

本实例实现一个使用挤压手势将气球挤扁的效果。运行效果如图 9.6 所示。

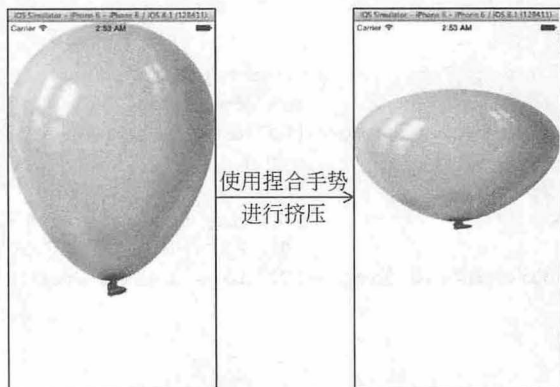


图 9.6 运行效果

【实现过程】

- (1) 创建一个项目，命名为“被挤扁的气球”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现实例变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface ViewController : UIViewController{
    CGFloat lastScale;
}
@end
```

- (4) 打开 ViewController.m 文件，编写代码，实现使用捏合手势将气球挤扁。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //创建并添加空白视图对象
    UIView *view=[[UIView alloc] initWithFrame:CGRectMake (-5,0,image.size.
width,image.size.height)];
    [self.view addSubview:view]; //添加视图对象
    UIImage *image=[UIImage imageNamed:@"1.png"];
    UIImageView *im=[[UIImageView alloc] initWithFrame:[view frame]];
    [im setImage:image]; //设置图像
    [view addSubview:im];
    UIPinchGestureRecognizer *pinchRecognizer = [[UIPinchGestureRecognizer
alloc] initWithTarget:self action:@selector(scale:)];
    //创建 UIPinchGestureRecognizer 识别器
    [view addGestureRecognizer:pinchRecognizer]; //在视图上添加识别器
}
//实现挤压的功能
-(void)scale:(id)sender {
    [self.view bringSubviewToFront:[(UIPinchGestureRecognizer*)sender view]];
    //当手指离开屏幕时，将 lastscale 设置为 1.0
    if([(UIPinchGestureRecognizer*)sender state] == UIGestureRecognizerStateEnded) {
        lastScale = 1.0;
        return;
    }
    //实现挤压
    CGFloat scale = 1.0 - (lastScale - [(UIPinchGestureRecognizer*)sender
scale]); //设置缩放值
    CGAffineTransform currentTransform = [(UIPinchGestureRecognizer*)
sender view].transform;
    CGAffineTransform newTransform = CGAffineTransformScale(currentTransform,
1.0, scale); //缩放
    [(UIPinchGestureRecognizer*)sender view] setTransform:newTransform;
    //设置改变
    lastScale = [(UIPinchGestureRecognizer*)sender scale];
}
//获取是否有 UIPanGestureRecognizer
- (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer
*)otherGestureRecognizer {
    return ![gestureRecognizer isKindOfClass:[UIPanGestureRecognizer class]];
}
```

【代码解析】

本实例关键功能是挤压气球的手势以及气球的形状改变。下面依次讲解这两个知识点。

1. 挤压气球的手势

UIPinchGestureRecognizer 手势识别器适用于缩放的手势。本实例就是使用了此手势识别器实现了挤压的手势。代码如下：

```
UIPinchGestureRecognizer *pinchRecognizer = [[UIPinchGestureRecognizer
alloc] initWithTarget:self action:@selector(scale:)];
[view addGestureRecognizer:pinchRecognizer];
```

2. 气球的形状改变

气球形状的改变是通过 setTransform:方法实现的，它的功能就是改变视图的形状。在本实例中的代码如下：

```
CGAffineTransform newTransform = CGAffineTransformScale(currentTransform,
1.0, scale);
[[UIPinchGestureRecognizer*]sender view] setTransform:newTransform];
```

实例 99 撕裂图像

【实例描述】

本实例的功能是使用捏合的手势实现撕裂图像的效果。当用户在使用向外的捏合手势时，图像被撕裂为两部分。运行效果如图 9.7 所示。

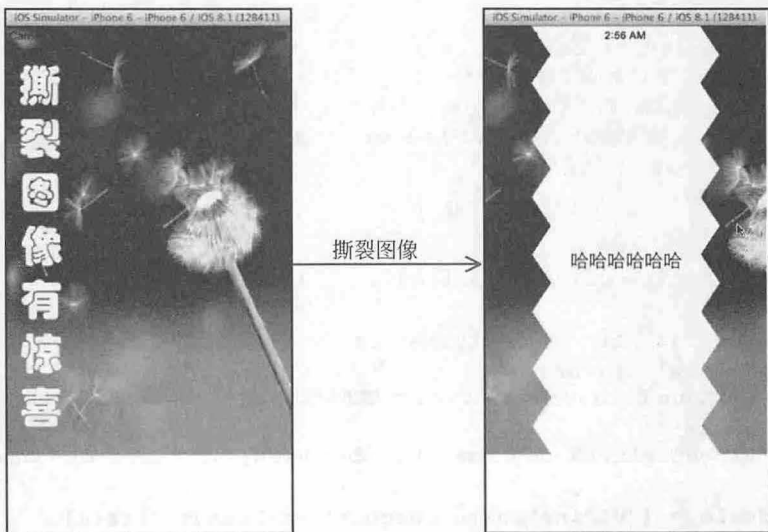


图 9.7 运行效果

【实现过程】

- (1) 创建一个项目，命名为“撕裂图像”。
- (2) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。

(3) 创建一个基于 UIImage 类的分类 SplitImageIntoTwoParts。

(4) 打开 UIImage+SplitImageIntoTwoParts.h 文件, 编写代码, 实现宏定义、方法的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#define SAWTOOTH_COUNT 15
#define SAWTOOTH_WIDTH_FACTOR 30
@interface UIImage (SplitImageIntoTwoParts)
+ (NSArray*)splitImageIntoTwoParts:(UIImage*)image;
@end
```

(5) 打开 UIImage+SplitImageIntoTwoParts.m 文件, 编写代码, 实现将图像分割为两部分。程序代码如下:

```
+ (NSArray *)splitImageIntoTwoParts:(UIImage *)image{
    CGFloat scale = [[UIScreen mainScreen] scale];
    NSMutableArray *array = [NSMutableArray arrayWithCapacity:2];
                                                                    //创建可变数组

    CGFloat width,height,widthgap,heightgap;
    int piceCount = SAWTOOTH_COUNT;
    width = image.size.width;
    height = image.size.height;
    widthgap = width/SAWTOOTH_WIDTH_FACTOR;
    heightgap = height/piceCount;
    CGContextRef context;
    CGImageRef imageMasked;
    UIImage *leftImage,*rightImage;
    //第一部分
    UIGraphicsBeginImageContext(CGSizeMake(width*scale, height*scale));
                                                                    //创建位图的上下文
    context = UIGraphicsGetCurrentContext();
                                                                    //创建图形上下文
    CGContextScaleCTM(context, scale, scale);
                                                                    //缩放
    CGContextMoveToPoint(context, 0, 0);
                                                                    //设置开始点
    int a=-1;
    //循环
    for (int i=0; i<piceCount+1; i++) {
        CGContextAddLineToPoint(context, width/2+(widthgap*a), heightgap*i);
                                                                    //添加结束点
        a= a*-1;
    }
    CGContextAddLineToPoint(context, 0, height);
    CGContextClosePath(context);
                                                                    //关闭路径
    CGContextClip(context);
    [image drawAtPoint:CGPointMake(0, 0)];
    imageMasked = CGBitmapContextCreateImage(context);
                                                                    //创建位图上下文
    leftImage = [UIImage imageWithCGImage:imageMasked scale:scale orientation:
    UIImageOrientationUp];
    [array addObject:leftImage];
                                                                    //添加对象
    UIGraphicsEndImageContext();
    //第二部分
    UIGraphicsBeginImageContext(CGSizeMake(width*scale, height*scale));
    context = UIGraphicsGetCurrentContext();
    CGContextScaleCTM(context, scale, scale);
                                                                    //缩放
    CGContextMoveToPoint(context, width, 0);
    .....
    UIGraphicsEndImageContext();
    return array;
}
```

}

- (6) 创建一个基于 UIView 类的 View 类。
- (7) 打开 View.h 文件，编写代码，实现头文件、属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "UIImage+SplitImageIntoTwoParts.h"
@interface View : UIView
//属性
@property (nonatomic,strong) UIImageView *image;
@property (nonatomic) BOOL isSplitting;
@property (nonatomic,strong) UIImageView *left;
@property (nonatomic,strong) UIImageView *right;
@end
```

- (8) 打开 View.m 文件。编写代码，实现将图像撕裂的效果。使用的方法如表 9-5 所示。

表 9-5 View.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
pinch:	实现使用捏合的手势将图像撕开
animationDidStop:finished:context:	在动画结束后调用

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，initWithFrame:方法实现初始化的功能即创建图像视图以及手势识别器对象等。程序代码如下：

```
-(id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.image = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"1.jpg"]];
        [self addSubview:self.image];
        //创建捏合的手势识别器对象
        UIPinchGestureRecognizer *pinchRecognizer = [[UIPinchGestureRecognizer alloc] initWithTarget:self action:@selector(pinch:)];
        [self addGestureRecognizer:pinchRecognizer];
    }
    return self;
}
```

pinch:方法实现使用捏合的手势将图像撕为两块。程序代码如下：

```
-(void)pinch:(UIPinchGestureRecognizer*)recognizer
{
    //判断手势识别器的状态
    if (recognizer.state == UIGestureRecognizerStateBegan) {
        //判断 scale 是否大于 1
        if (recognizer.scale>1) {
            self.isSplitting = YES;
            NSArray *array = [UIImage splitImageIntoTwoParts:self.image.image];
            //实例化视图对象
            self.left = [[UIImageView alloc] initWithImage:[array objectAtIndex:0]];
            self.right = [[UIImageView alloc] initWithImage:[array objectAtIndex:1]];
            [self addSubview:self.left]; //添加视图对象
            [self addSubview:self.right];
            [self.image setHidden:YES]; //隐藏图像
        }
    }
}
```



```

    }else{
        self.isSplitting = NO;
    }
}
}else if (recognizer.state == UIGestureRecognizerStateChanged) {
    if (self.isSplitting) {
        //改变图像的位置
        self.left.transform = CGAffineTransformMakeTranslation(-160*
(recognizer.scale-1), 0);
        self.right.transform = CGAffineTransformMakeTranslation(160*
(recognizer.scale-1), 0);
    }
}
}else if (recognizer.state == UIGestureRecognizerStateEnded){
    //当手势识别器结束后的动画效果
    [UIView beginAnimations:@"split" context:nil];
    [UIView setAnimationDelegate:self]; //设置动画委托
    [UIView setAnimationDidStopSelector:@selector(animationDidStop:
finished:context:)];
    self.left.transform = CGAffineTransformIdentity; //设置改变
    self.right.transform = CGAffineTransformIdentity;
    [UIView commitAnimations];
}
}
}

```

(9) 打开 Main.storyboard 文件，在设计界面中添加 Label 标签控件。将此控件的 Text 设置为“哈哈哈哈哈”，将 Font 设置为 System 21.0，将 Alignment 设置为居中。

(10) 打开 ViewController.h 文件，编写代码，实现头文件以及对象的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "View.h"
@interface ViewController : UIViewController{
    View *vv;
}
@end

```

(11) 打开 ViewController.m 文件，编写代码。实现 vv 对象的初始化以及添加。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    View *view = [[View alloc] initWithFrame:self.view.frame]; //实例化对象
    [self.view addSubview:view];
}

```

【代码解析】

本实例关键功能是图像的裂痕以及撕裂图像的效果。下面依次讲解这两个知识点。

1. 图像的裂痕

在本实例中图像的裂痕是通过 CGContextMoveToPoint 和 CGContextAddLineToPoint 函数通过绘制直线实现的。以绘制第一部分的裂痕为例，代码如下：

```

CGContextMoveToPoint(context, 0, 0);
int a=-1;
for (int i=0; i<piceCount+1; i++) {

```

```
CGContextAddLineToPoint(context, width/2+(widthgap*a), heightgap*i);
a= a*-1;
}
CGContextAddLineToPoint(context, 0, height);
```

2. 撕裂图像的效果

在本实例中撕裂图像是通过使用 `UIPinchGestureRecognizer` 手势识别器实现的。使用此手势识别器实时，会调用 `pinch:` 方法。在此方法首先会判断手势识别器的状态，如果为 `UIGestureRecognizerStateBegan` 状态，并且缩放大于 1 时，将图像分为左右两个部分。代码如下：

```
self.isSplitting = YES;
NSArray *array = [UIImage splitImageIntoTwoParts:self.image.image];
self.left = [[UIImageView alloc] initWithImage:[array objectAtIndex:0]];
self.right = [[UIImageView alloc] initWithImage:[array objectAtIndex:1]];
self addSubview:self.left;
[self addSubview:self.right];
[self.image setHidden:YES];
```

当手势的状态为 `UIGestureRecognizerStateChanged` 时，就将左右两个图像分开，实现图像的移动。代码如下：

```
self.left.transform = CGAffineTransformMakeTranslation(-160*(recognizer.
scale-1), 0);
self.right.transform = CGAffineTransformMakeTranslation(160*(recognizer.
scale-1), 0);
```

实例 100 一个手指实现缩放

【实例描述】

在 iOS 中进行图像的缩放需要使用两个手指才可以实现。在本实例中就打破它的理念，仅采用一个手指就可以将图像进行任意的缩放了。当用户向上滑动时，图像放大；当用户向下滑动时，图像就缩小了。运行效果如图 9.8 所示。

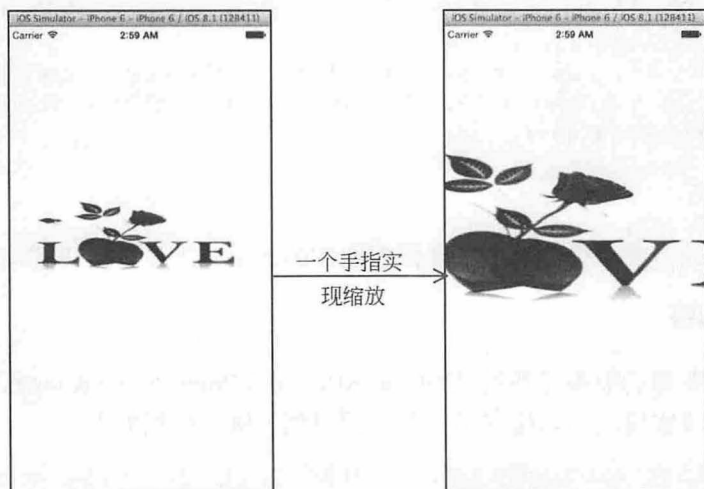


图 9.8 运行效果

【实现过程】

- (1) 创建一个项目，命名为“一个手指实现缩放”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIGestureRecognizer 类的 ZoomGestureRecognizer 类。
- (4) 打开 ZoomGestureRecognizer.h 文件，编写代码，实现头文件以及属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <UIKit/UIGestureRecognizerSubclass.h>
@interface ZoomGestureRecognizer : UIGestureRecognizer
//属性
@property (nonatomic, assign) NSUInteger numberOfTapsRequired;
.....
@property (nonatomic, assign) CGFloat initialScale;
@property (nonatomic, assign) BOOL pressedLongEnough;
@end
```

- (5) 打开 ZoomGestureRecognizer.m 文件，编写代码，实现使用一个手指进行缩放的手势识别器。使用的方法如表 9-6 所示。

表 9-6 ZoomGestureRecognizer.m 文件中方法总结

方 法	功 能
initWithTarget:action:	初始化
screenLocationYOfTouch:	获取触摸的位置 y
zoomRubberBandScaleForZoomScale:minimumZoomScale:maximumZoomScale:	获取缩放
locationInView:	获取手势在视图中的位置
handleTimer:	对定时器的处理
reset	重新设置
touchesBegan:withEvent:	开始触摸
touchesMoved:withEvent:	移动触摸
touchesEnded:withEvent:	结束触摸
touchesCancelled:withEvent:	触摸取消

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，initWithTarget:action:方法实现了初始化的功能。程序代码如下：

```
- (instancetype)initWithTarget:(id)target action:(SEL)action {
    self = [super initWithTarget:target action:action];
    if (!self) return nil;
    _numberOfTapsRequired = 1; //设置手指的个数
    _allowableMovement = 10.0f;
    _scaleFactor = 1.5f;
    _scale = 1.0f;
    _minimumScale = 1.0f; //设置缩放的最小值
    _maximumScale = 1.0f; //设置缩放的最大值
    return self;
}
```

zoomRubberBandScaleForZoomScale:minimumZoomScale:maximumZoomScale: 方法对缩放的值进行获取。程序代码如下：


```

- (CGFloat)zoomRubberBandScaleForZoomScale:(CGFloat)scale minimumZoomScale:
(CGFloat)minimumZoomScale maximumZoomScale:(CGFloat)maximumZoomScale {
    //判断缩放是否小于等于缩放的最大值
    if (scale <= maximumZoomScale) {
        //判断缩放是否大于等于缩放的最小值
        if (scale >= minimumZoomScale)
            return scale;
        else
            scale = minimumZoomScale / (2.0f - (scale / minimumZoomScale));
        //设置缩放
    } else
        scale = maximumZoomScale * (2.0f - (maximumZoomScale / scale));
    //设置缩放
    return scale;
}

```

`touchesBegan:withEvent:`方法在开始触摸时调用，实现判断手势识别的状态以及对定时器的设置。程序代码如下：

```

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesBegan:touches withEvent:event];
    UITouch *touch = [touches anyObject];
    if (self.state == UIGestureRecognizerStatePossible && [touches count]
    == 1 && touch.tapCount == self.numberOfTapsRequired + 1) {
        //创建并设置定时器
        NSTimer *timer = [NSTimer timerWithTimeInterval:self.minimumPressDuration
        target:self selector:@selector(handleTimer:) userInfo:nil repeats:NO];
        [[NSRunLoop currentRunLoop] addTimer:timer forMode:NSRunLoopCommonModes];
        self.timer = timer;
        self.initialLocationOnScreen = [touch.window convertPoint:[touch
        locationInView:nil] toWindow:nil];
    } else if (self.state != UIGestureRecognizerStateFailed)
        self.state = UIGestureRecognizerStateCancelled;
    //设置状态
}

```

`touchesMoved:withEvent:`方法在触摸移动时调用，实现放大缩小的功能。程序代码如下：

```

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    [super touchesMoved:touches withEvent:event];
    //判断状态是否不等于 UIGestureRecognizerStatePossible、UIGestureRecognizer-
    StateBegan、UIGestureRecognizerStateChanged
    if (self.state != UIGestureRecognizerStatePossible &&
        self.state != UIGestureRecognizerStateBegan &&
        self.state != UIGestureRecognizerStateChanged) return;
    UITouch *touch = [touches anyObject];
    if (touch.tapCount != self.numberOfTapsRequired + 1) return;
    //获取触摸位置
    CGPoint currentLocationOnScreen = [touch.window convertPoint:[touch
    locationInView:nil] toWindow:nil];
    //判断 state 是否等于 UIGestureRecognizerStatePossible
    if (self.state == UIGestureRecognizerStatePossible) {
        if (self.pressedLongEnough)
            self.state = UIGestureRecognizerStateBegan;
        //设置状态
        else {
            CGFloat movedDistance = sqrtf(powf(currentLocationOnScreen.x -
            self.initialLocationOnScreen.x, 2.0f) + powf(currentLocationOnScreen.

```

```

        y - self.initialLocationOnScreen.y, 2.0f)); //获取移动的距离
        if (movedDistance > self.allowableMovement)
            self.state = UIGestureRecognizerStateFailed; //设置状态
    }
} else if (self.state == UIGestureRecognizerStateBegan) {
    self.initialScale = self.scale;
    self.state = UIGestureRecognizerStateChanged; //设置状态
} else if (self.state == UIGestureRecognizerStateChanged) {
    //获取距离
    CGFloat movedDistance = self.initialLocationOnScreen.y - current-
        LocationOnScreen.y;
    CGFloat movedPercent = ABS(movedDistance) / [UIScreen mainScreen].
        bounds.size.height;
    CGFloat scalePercent = 1.0f + (self.scaleFactor - 1.0f) * movedPercent;
    CGFloat newScale = self.initialScale * (movedDistance > 0.0f ?
        scalePercent : 1.0f / scalePercent); //获取新的缩放值
    if (self.bouncesScale){
        self.scale = [self zoomRubberBandScaleForZoomScale:newScale
            minimumZoomScale:self.minimumScale maximumZoomScale:self.
            maximumScale]; //设置缩放
    }else{
        self.scale = newScale;
    }
}
}
}

```

(6) 打开 ViewController.h 文件, 编写代码, 实现头文件、遵守协议、插座变量以及对象的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "ZoomGestureRecognizer.h"
@interface ViewController : UIViewController<UIGestureRecognizerDelegate>{
    IBOutlet UIImageView *imageView;
    ZoomGestureRecognizer *fingerZoom;
}
@end

```

(7) 打开 Main.storyboard 文件, 从视图库中拖动 Image View 图像视图到设计界面中, 将此视图的 Image 设置为 1.png。将插座变量 imageView 与此视图关联。

(8) 打开 ViewController.m 文件, 编写代码, 实现使用一个手指进行缩放的手势识别器对象的初始化以及缩放功能。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    fingerZoom = [[ZoomGestureRecognizer alloc] initWithTarget:self action:
        @selector(handleFingerZoom:)]; //创建手势识别器对象
    fingerZoom.numberOfTapsRequired = 0;
    fingerZoom.scaleFactor = 1.1; //设置缩放因子
    [self.view addGestureRecognizer:fingerZoom];
}

- (void)handleFingerZoom:(ZoomGestureRecognizer *)gestureRecognizer {
    imageView.transform = CGAffineTransformScale(imageView.transform,
        gestureRecognizer.scale, gestureRecognizer.scale); //缩放
}

```



```
gestureRecognizer.scale = 1;
}
```

【代码解析】

本实例关键功能是图像的放大缩小。下面就是这个知识点的详细讲解。

在本实例中使用一个手指进行图像的放大缩小，使用的其实是触摸事件，在 `touchesMoved:withEvent:` 方法中实现当触摸移动时，实现图像的放大缩小。其中，首先需要对当前的状态进行判断，从而实现放大缩小的功能。代码如下：

```
//判断当前的状态是否为 UIGestureRecognizerStatePossible
if (self.state == UIGestureRecognizerStatePossible) {
    if (self.pressedLongEnough)
        self.state = UIGestureRecognizerStateBegan;
    else {
        CGFloat movedDistance = sqrtf(powf(currentLocationOnScreen.x -
            self.initialLocationOnScreen.x, 2.0f) + powf(currentLocationOnScreen.
            y - self.initialLocationOnScreen.y, 2.0f));
        if (movedDistance > self.allowableMovement)
            self.state = UIGestureRecognizerStateFailed;
    }
}
//判断当前的状态是否为 UIGestureRecognizerStateBegan
} else if (self.state == UIGestureRecognizerStateBegan) {
    self.initialScale = self.scale;
    self.state = UIGestureRecognizerStateChanged;
}
//判断当前的状态是否为 UIGestureRecognizerStateChanged
} else if (self.state == UIGestureRecognizerStateChanged) {
    CGFloat movedDistance = self.initialLocationOnScreen.y - current-
        LocationOnScreen.y;
    CGFloat movedPercent = ABS(movedDistance) / [UIScreen mainScreen].
        bounds.size.height;
    CGFloat scalePercent = 1.0f + (self.scaleFactor - 1.0f) * movedPercent;
    CGFloat newScale = self.initialScale * (movedDistance > 0.0f ?
        scalePercent : 1.0f / scalePercent);
    if (self.bouncesScale) {
        self.scale = [self zoomRubberBandScaleForZoomScale:newScale minimumZoomScale:
            self.minimumScale maximumZoomScale:self.maximumScale];
    } else {
        self.scale = newScale;
    }
}
```

实例 101 仿小米手机的解锁功能

【实例描述】

本实例实现的是小米手机的解锁功能。当用户将拖曳按钮拖动到外圆的线上时，在标签中就会显示 Unlocked，表示解锁成功；如果用户没有将拖曳按钮拖动到外圆的线上时，就是在标签中显示 Did not unlocked，表示没有解锁成功。运行效果如图 9.9 所示。

【实现过程】

- (1) 创建一个项目，命名为“仿小米手机的解锁功能”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。

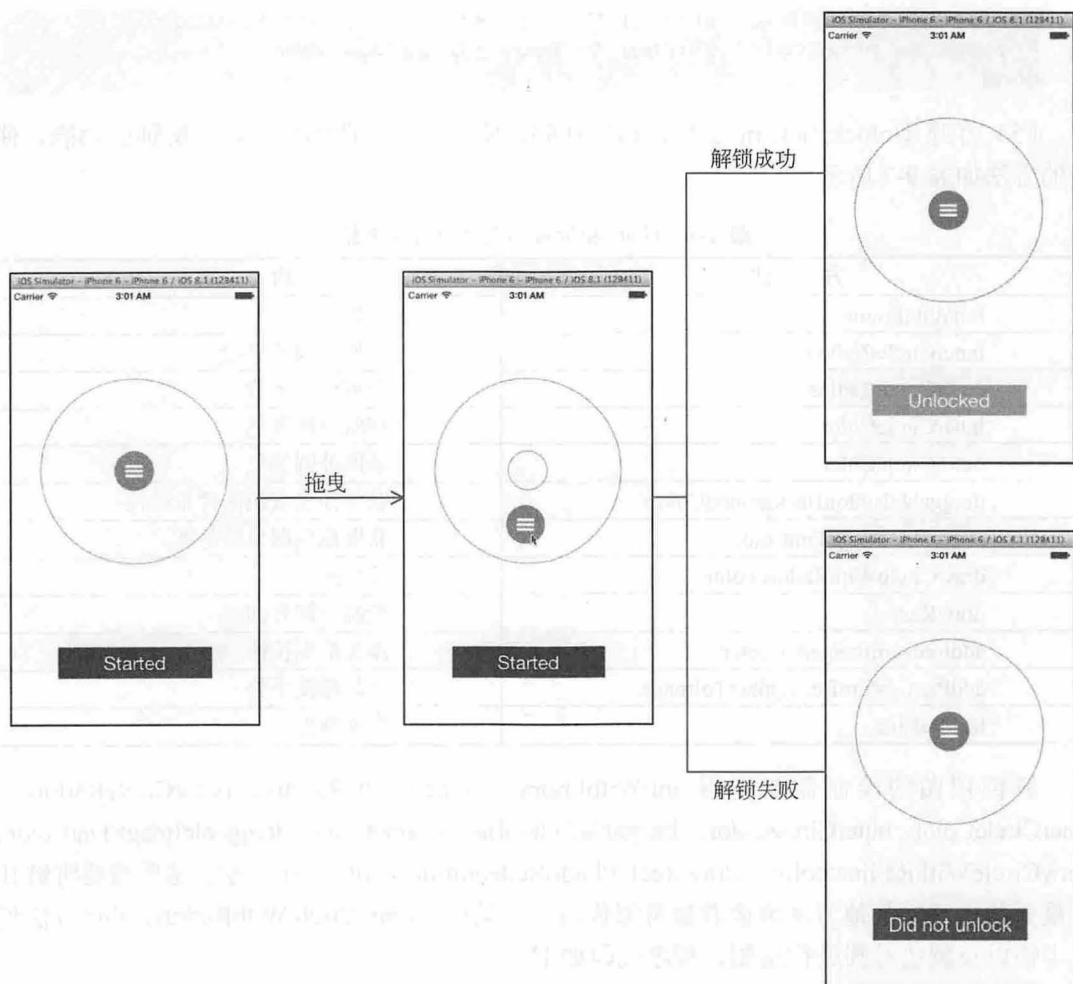


图 9.9 运行效果

(3) 创建一个基于 UIView 类的 UnlockView 类。

(4) 打开 UnlockView.h 文件，编写代码，实现头文件、宏定义、协议、实例变量以及属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@class UnlockView;
#define kDefaultInnerCircleRadius 25.0f;
//协议
@protocol UnlockViewDelegate <NSObject>
@optional
- (void)slideOutToUnlockViewDidStartToDrag:(UnlockView *)slideOutToUnlockView;
- (void)slideOutToUnlockViewDidUnlock:(UnlockView *)slideOutToUnlockView;
- (void)slideOutToUnlockViewDidNotUnlock:(UnlockView *)slideOutToUnlockView;
@end
@interface UnlockView : UIView{
    BOOL _unlockOnRelease;
}
//属性
@property (nonatomic, strong) UIButton *dragButton;
.....
```

```
@property (nonatomic, strong) UIColor *draggableImageTintColor;
@property (nonatomic, strong) UIImage *draggableImage;
@end
```

(5) 打开 UnlockView.m 文件，编写代码，实现解锁视图的绘制以及解锁的功能。使用的方法如表 9-7 所示。

表 9-7 UnlockView.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
innerCircleRadius	获取内圆的半径
outerCircleRadius	获取外圆半径
innerCircleColor	获取内圆颜色
outerCircleColor	获取外圆颜色
draggableButtonBackgroundColor	获取拖曳按钮的背景颜色
draggableImageTintColor	获取拖曳图像的色泽
drawCircleWithRadius:color:	绘制圆
drawRect:	绘制内圆外圆
addRedeemImageAtCenter	添加拖曳按钮
addPanGestureRecognizerToImage	添加拖曳手势
handlePan:	实现拖曳

解锁视图的绘制需要使用 initWithFrame:、innerCircleRadius、outerCircleRadius、innerCircleColor、outerCircleColor、draggableButtonBackgroundColor、draggableImageTintColor、drawCircleWithRadius:color:、drawRect:和 addRedeemImageAtCenter 方法。这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，drawCircleWithRadius:color:方法使用半径以及颜色对圆进行绘制。程序代码如下：

```
- (void)drawCircleWithRadius:(CGFloat)radius color:(UIColor *)color
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(context, 1.0); //设置线宽
    CGContextSetStrokeColorWithColor(context, color.CGColor);
    CGRect circle = CGRectMake(self.center.x - radius, self.center.y -
    radius, 2 * radius, 2 * radius);
    CGContextAddEllipseInRect(context, circle); //添加圆
    CGContextStrokePath(context);
}
```

drawRect:方法实现对内圆外圆的绘制等功能。程序代码如下：

```
- (void)drawRect:(CGRect)rect
{
    [self drawCircleWithRadius:self.innerCircleRadius color:self.innerCircleColor]; //绘制内圆
    [self drawCircleWithRadius:self.outerCircleRadius color:self.outerCircleColor]; //绘制外圆

    [self addRedeemImageAtCenter];
    [self addPanGestureRecognizerToImage];
}
```

addRedeemImageAtCenter 方法是对拖曳按钮的添加。程序代码如下：

```

- (void)addRedeemImageAtCenter
{
    //创建并设置按钮对象
    _dragButton = [UIButton buttonWithType:UIButtonTypeCustom];
    _dragButton.frame = CGRectMake(0, 0, 2*self.innerCircleRadius, 2*self.
innerCircleRadius); //设置框架
    _dragButton.backgroundColor = self.draggableButtonBackgroundColor; //设置背景颜色
    _dragButton.tintColor = self.draggableImageTintColor;
    _dragButton.layer.cornerRadius = self.innerCircleRadius; //设置圆角半径
    _dragButton.clipsToBounds = YES;
    UIImage *image = _draggableImage ? [UIImage imageNamed:@"1.png"];
    //判断是否实现了 imageWithRenderingMode:方法
    if ([image respondsToSelector:@selector(imageWithRenderingMode:)]) {
        image = [image imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate];
        [_dragButton setImage:image forState:UIControlStateNormal];
        [self addSubview:_dragButton];
        _dragButton.center = self.center; //设置中心位置
    }
}

```

解锁的功能需要使用 `addPanGestureRecognizerToImage` 和 `handlePan:` 方法实现。其中, `handlePan:` 方法实现拖曳功能以及解锁功能。程序代码如下

```

- (void)handlePan:(UIPanGestureRecognizer *)gestureRecognizer
{
    //判断 state 是否为 UIGestureRecognizerStateBegan
    if (gestureRecognizer.state == UIGestureRecognizerStateBegan) {
        //判断是否实现了 slideOutToUnlockViewDidStartToDrag:方法
        if ([_delegate respondsToSelector:@selector(slideOutToUnlockView
DidStartToDrag:)]) {
            [_delegate slideOutToUnlockViewDidStartToDrag:self];
        }
    }
    //判断 state 是否为 UIGestureRecognizerStateChanged
    if (gestureRecognizer.state == UIGestureRecognizerStateChanged) {
        CGPoint translation = [gestureRecognizer translationInView:self]; //获取触摸
        if (abs(translation.x) <= self.outerCircleRadius && abs(translation.y)
<= self.outerCircleRadius) {
            //设置拖曳按钮的中心
            _dragButton.center = CGPointMake(self.center.x + translation.x,
self.center.y + translation.y);
        }
        _unlockOnRelease = sqrt(pow(_dragButton.center.x - self.center.x, 2) +
pow(_dragButton.center.y - self.center.y, 2)) > self.outerCircleRadius -
self.innerCircleRadius;
    }
    //判断 state 是否为 UIGestureRecognizerStateEnded
    if (gestureRecognizer.state == UIGestureRecognizerStateEnded) {
        if (_unlockOnRelease) {
            //解锁
            if ([self.delegate respondsToSelector:@selector(slideOutTo
UnlockViewDidUnlock:)]) {
                [self.delegate slideOutToUnlockViewDidUnlock:self];
            }
        } else {
            //没有解锁
            if ([self.delegate respondsToSelector:@selector(slideOutTo
UnlockViewDidNotUnlock:)]) {
                [self.delegate slideOutToUnlockViewDidNotUnlock:self];
            }
        }
    }
}

```

```
    }
    //动画效果
    [UIView animateWithDuration:0.2f animations:^(
        _dragButton.center = self.center;           //设置中心位置
    )];
}
```

（6）打开 ViewController.h 文件，编写代码，实现头文件、宏定义、遵守协议以及插座变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "UnlockView.h"
#define kHorizontalMargin 20.0f;
@interface ViewController : UIViewController<UnlockViewDelegate>{
    //插座变量
    IBOutlet UIView *containerView;
    IBOutlet UILabel *currentStateLabel;
}
@end
```

（7）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 9.10 所示。

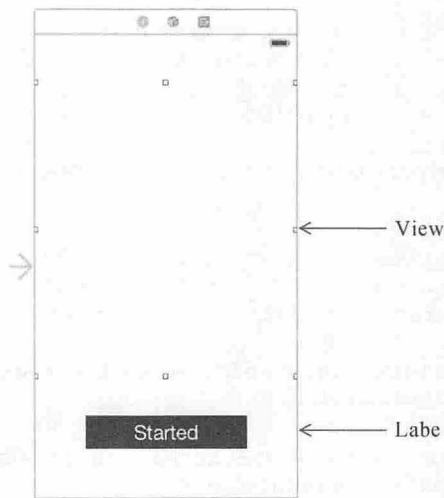


图 9.10 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 9-8 所示。

表 9-8 视图、控件设置

视图、控件	属 性 设 置	其 他
View		与插座变量 containerView 关联
Label	Text: Started Color: 白色 Font: System 25.0 Alignment: 居中 Background: 黑色	与插座变量 currentStateLabel 关联

(8) 打开 ViewController.m 文件, 编写代码, 实现将解锁视图添加到界面中, 以及一些协议的实现。程序代码如下:

```
//视图加载后调用, 实现初始化
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    UnlockView *unlockView = [[UnlockView alloc] initWithFrame:container
    View.bounds]; //实例化对象
    unlockView.delegate = self; //设置委托
    unlockView.outerCircleRadius = CGRectGetWidth(containerView.bounds) /
    2.0f - 2*kHorizontalMargin;
    [containerView addSubview:unlockView]; //添加视图对象
}
//开始拖曳按钮时调用
- (void)slideOutToUnlockViewDidStartToDrag:(UnlockView *)slideOutToUnlockView
{
    currentStateLabel.text = @"Started"; //设置文本内容
    currentStateLabel.backgroundColor = [UIColor blackColor];
}
//解锁后调用
- (void)slideOutToUnlockViewDidUnlock:(UnlockView *)slideOutToUnlockView
{
    currentStateLabel.text = @"Unlocked";
    currentStateLabel.backgroundColor = [UIColor colorWithRed:0 green:0.7
    blue:0 alpha:1]; //设备背景颜色
}
//没有解锁后调用
- (void)slideOutToUnlockViewDidNotUnlock:(UnlockView
*)slideOutToUnlockView
{
    currentStateLabel.text = @"Did not unlock"; //设置文本内容
    currentStateLabel.backgroundColor = [UIColor colorWithRed:0.7 green:0
    blue:0 alpha:1];
}
```

【代码解析】

本实例关键功能是拖曳按钮的拖曳, 以及解锁的判断。下面依次讲解这两个知识点。

1. 拖曳按钮的拖曳

在本实例中拖曳按钮的拖曳使用的是 UIPanGestureRecognizer 拖曳手势识别器实现的, 代码如下:

```
UIPanGestureRecognizer *panGesture = [[UIPanGestureRecognizer alloc]
initWithTarget:self action:@selector(handlePan:)];
[_dragButton addGestureRecognizer:panGesture];
```

2. 解锁的判断

对于解锁的判断, 本实例采用的是拖曳按钮中心位置的偏移量和外圆到内圆的距离比较实现的, 代码如下:

```
_unlockOnRelease = sqrt(pow(_dragButton.center.x-self.center.x, 2) +
pow(_dragButton.center.y-self.center.y, 2)) > self.outerCircleRadius -
self.innerCircleRadius;
```

如果拖曳按钮中心位置的偏移量大于外圆到内圆的距离，表示解锁成功；如果拖曳按钮中心位置的偏移量小于外圆到内圆的距离，表示解锁失败。程序代码如下：

```
if (_unlockOnRelease) {
    if ([self.delegate respondsToSelector:@selector(slideOutToUnlockViewDidUnlock:)])
        [self.delegate slideOutToUnlockViewDidUnlock:self];
} else {
    if ([self.delegate respondsToSelector:@selector(slideOutToUnlockViewDidNotUnlock:)])
        [self.delegate slideOutToUnlockViewDidNotUnlock:self];
}
```

实例 102 QQ 的解锁功能

【实例描述】

QQ 的手机解锁功能是一种很新颖的解锁方式，它采用了让用户绘制一种手势的解锁功能，增强了 QQ 的安全保障。在本实例中就为各位读者实现 QQ 解锁的功能。当用户在解锁界面输入手势后，会在标签栏中自动转换为数字密码。运行效果如图 9.11 所示。

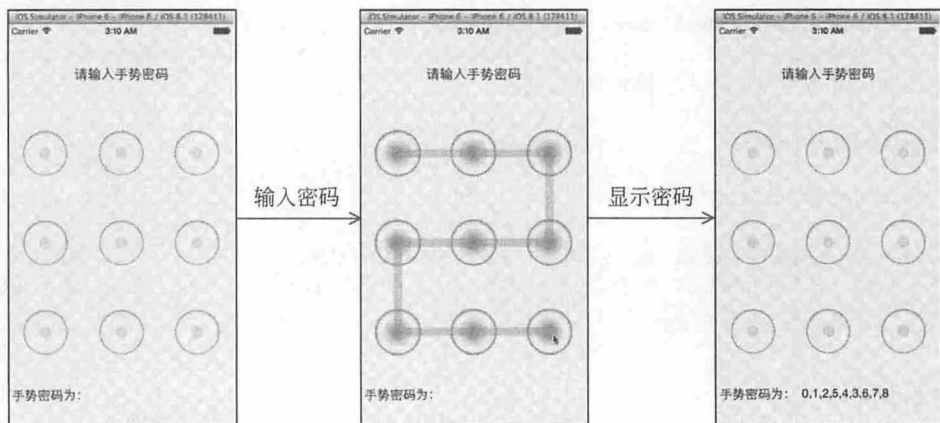


图 9.11 运行效果

【实现过程】

- (1) 创建一个项目，命名为“QQ 的解锁功能”。
- (2) 添加图像 1.png、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 UIView 类的 GestureLockView 类。
- (4) 打开 GestureLockView.h 文件，编写代码，实现头文件、数据结构的定义、协议、属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@class GestureLockView;
//定义数据结构
struct {
    unsigned int didBeginWithPasscode : 1;
    unsigned int didEndWithPasscode : 1;
    unsigned int didCanceled : 1;
```

```

} _delegateFlags;
//协议
@protocol GestureLockViewDelegate <NSObject>
@optional
//开始解锁时调用
- (void)gestureLockView:(GestureLockView *)gestureLockView didBeginWith
Passcode:(NSString *)passcode;
//解锁结束后调用
- (void)gestureLockView:(GestureLockView *)gestureLockView didEndWith
Passcode:(NSString *)passcode;
- (void)gestureLockView:(GestureLockView *)gestureLockView didCanceled
WithPasscode:(NSString *)passcode;
@end
@interface GestureLockView : UIView
//属性
@property (nonatomic, strong) UIView *contentView;
.....
@property (nonatomic, assign) UIEdgeInsets contentInsets;
@property (nonatomic, weak) id<GestureLockViewDelegate> delegate;
@end

```

(5) 打开 GestureLockView.m 文件, 编写代码, 实现解锁视图的绘制以及触摸功能即实现解锁。使用的方法如表 9-9 所示。

表 9-9 GestureLockView.m文件中方法总结

方 法	功 能
imageWithColor:size:	获取图像
_buttonContainsThePoint:	获取按钮包含的点
_lockViewInitialize	解锁视图的初始化
initWithFrame:	通过框架进行初始化
initWithCoder:	初始化实例对象
layoutSubviews	布局视图
drawRect:	绘制贝塞尔曲线
touchesBegan:withEvent:	开始触摸
touchesMoved:withEvent:	移动触摸
touchesEnded:withEvent:	结束触摸
touchesCancelled:withEvent:	取消触摸
setNormalGestureNodeImage:	设置手势节点图像
setSelectedGestureNodeImage:	设置被选择后的手势节点图像
setDelegate:	设置委托
setNumberOfGestureNodes:	设置手势节点的数目

解锁视图的绘制需要使用 imageWithColor:size:、_lockViewInitialize、initWithFrame:、initWithCoder:、layoutSubviews、drawRect:、setNormalGestureNodeImage:、setSelectedGestureNodeImage:、setDelegate:、setNumberOfGestureNodes:方法实现。这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, _lockViewInitialize 方法实现对解锁视图的初始化。程序代码如下:

```

- (void)_lockViewInitialize{
    self.backgroundColor = [UIColor clearColor];           //设置背景颜色
    //对线的设置

```

```

self.lineColor = [[UIColor blackColor] colorWithAlphaComponent:0.3];
self.lineWidth = kLineDefaultWidth; //设置线宽
//对内容的设置
self.contentInsets = UIEdgeInsetsMake(0, 0, 0, 0);
self.contentView = [[UIView alloc] initWithFrame:UIEdgeInsets
InsetRect(self.bounds, self.contentInsets)];
self.contentView.backgroundColor = [UIColor clearColor]; //设备背景颜色
[self addSubview:self.contentView];
//对解锁视图中按钮的设置
self.buttonSize = CGSizeMake(kNodeDefaultWidth, kNodeDefaultHeight);
//设置按钮的尺寸

//设置普通的图像
self.normalGestureNodeImage = [self imageWithColor:[UIColor greenColor]
size:self.buttonSize];
//设置选择后的图像
self.selectedGestureNodeImage = [self imageWithColor:[UIColor redColor]
size:self.buttonSize];
self.numberOfGestureNodes = kNumberOfNodes;
self.gestureNodesPerRow = kNodesPerRow;
self.selectedButtons = [NSMutableArray array]; //设置选择的按钮
self.trackedLocationInContentView = CGPointMake(kTrackedLocation
InvalidInContentView, kTrackedLocationInvalidInContentView);
}

```

layoutSubviews 方法实现对解锁视图的布局。程序代码如下：

```

- (void)layoutSubviews{
    [super layoutSubviews];
    self.contentView.frame = UIEdgeInsetsInsetRect(self.bounds, self.
contentInsets); //设置框架
CGFloat horizontalNodeMargin = (self.contentView.bounds.size.width -
self.buttonSize.width * self.gestureNodesPerRow)/(self.gestureNodesPerRow - 1);
NSUInteger numberOfRows = ceilf((self.numberOfGestureNodes * 1.0 /
self.gestureNodesPerRow));
CGFloat verticalNodeMargin = (self.contentView.bounds.size.height - self.
buttonSize.height * numberOfRows)/(numberOfRows - 1);
//循环，对按钮的设置
for (int i = 0; i < self.numberOfGestureNodes ; i++) {
    int row = i / self.gestureNodesPerRow;
    int column = i % self.gestureNodesPerRow;
    UIButton *button = [self.buttons objectAtIndex:i]; //实例化按钮
    //设置框架
    button.frame = CGRectMake(floorf((self.buttonSize.width + horizon
talNodeMargin) * column), floorf((self.buttonSize.height + vertical
NodeMargin) * row), self.buttonSize.width, self.buttonSize.height);
}
}

```

drawRect:方法实现对贝塞尔曲线的绘制。程序代码如下：

```

- (void)drawRect:(CGRect)rect{
    [super drawRect:rect];
    //判断是否有选中的按钮
    if ([self.selectedButtons count] > 0) {
        UIBezierPath *bezierPath = [UIBezierPath bezierPath]; //创建贝塞尔路径
        UIButton *firstButton = [self.selectedButtons objectAtIndex:0];
        [bezierPath moveToPoint:[self convertPoint:firstButton.center
fromView:self.contentView]];
        //遍历选中的按钮
    }
}

```

```

for (int i = 1; i < [self.selectedButtons count]; i++) {
    UIButton *button = [self.selectedButtons objectAtIndex:i];
    //实例化按钮对象
    [bezierPath addLineToPoint:[self convertPoint:button.center
fromView:self.contentView]];
}
if (self.trackedLocationInContentView.x != kTrackedLocationInvalid
InContentView &&
    self.trackedLocationInContentView.y != kTrackedLocationInvalid
InContentView) {
    [bezierPath addLineToPoint:[self convertPoint:self.tracked
LocationInContentView fromView:self.contentView]]; //添加点
}
[bezierPath setLineWidth:self.lineWidth]; //设置线宽
[bezierPath setLineJoinStyle:kCGLineJoinRound]; //设置线的风格
[self.lineColor setStroke];
[bezierPath stroke];
}
}

```

setNormalGestureNodeImage:方法实现对手势中节点图像的设置,即对按钮的绘制。程序代码如下:

```

- (void)setNormalGestureNodeImage:(UIImage *)normalGestureNodeImage{
    //判断 _normalGestureNodeImage 是否不为 normalGestureNodeImage
    if (_normalGestureNodeImage != normalGestureNodeImage) {
        _normalGestureNodeImage = normalGestureNodeImage;
        CGSize buttonSize = self.buttonSize;
        buttonSize.width = self.buttonSize.width > normalGestureNodeImage.
size.width ? self.buttonSize.width : normalGestureNodeImage.size.
width; //设置宽度
        buttonSize.height = self.buttonSize.height > normalGestureNodeImage.
size.height ? self.buttonSize.height : normalGestureNodeImage.size.
height; //设置高度
        self.buttonSize = buttonSize;
        if (self.buttons != nil && [self.buttons count] > 0) {
            //为按钮设置图像
            for (UIButton *button in self.buttons) {
                [button setImage:normalGestureNodeImage forState:UIControlStateNormal];
            }
        }
    }
}

```

setNumberOfGestureNodes:方法实现对手势中节点数的设置。程序代码如下:

```

- (void)setNumberOfGestureNodes:(NSUInteger)numberOfGestureNodes{
    //判断 _normalGestureNodeImage 是否不为 normalGestureNodeImage
    if (_numberOfGestureNodes != numberOfGestureNodes) {
        _numberOfGestureNodes = numberOfGestureNodes;
        if (self.buttons != nil && [self.buttons count] > 0) {
            //遍历
            for (UIButton *button in self.buttons) {
                [button removeFromSuperview]; //移除指定的视图对象
            }
        }
        NSMutableArray *buttonArray = [NSMutableArray arrayWithCapacity:
numberOfGestureNodes];
    }
}

```



```

//遍历节点数
for (NSUInteger i = 0; i < numberOfGestureNodes; i++) {
    UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
    button.tag = i; //设置 tag 值
    button.userInteractionEnabled = NO;
    button.frame = CGRectMake(0, 0, self.buttonSize.width, self.buttonSize.height);
    button.backgroundColor = [UIColor clearColor]; //设置背景颜色
    if (self.normalGestureNodeImage != nil) {
        //设置图像
        [button setImage:self.normalGestureNodeImage forState:
        UIControlStateNormal];
    }
    if (self.selectedGestureNodeImage != nil) {
        //设置图像
        [button setImage:self.selectedGestureNodeImage forState:
        UIControlStateSelected];
    }
    [buttonArray addObject:button]; //添加按钮对象
    [self.contentView addSubview:button];
}
self.buttons = [buttonArray copy];
}
}

```

触摸功能即实现解锁，需要使用 `_buttonContainsThePoint:`、`touchesBegan:withEvent:`、`touchesMoved:withEvent:`、`touchesEnded:withEvent:`、`touchesCancelled:withEvent:` 方法。其中，`touchesBegan:withEvent:` 方法在触摸开始时调用。程序代码如下：

```

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch = [touches anyObject];
    CGPoint locationInContentView = [touch locationInView:self.contentView]; //获取触摸位置
    UIButton *touchedButton = [self _buttonContainsThePoint:locationInContentView];
    //判断触摸的按钮是否不为空
    if (touchedButton != nil) {
        touchedButton.selected = YES;
        [self.selectedButtons addObject:touchedButton]; //添加按钮对象
        self.trackedLocationInContentView = locationInContentView;
        if (_delegateFlags.didBeginWithPasscode) {
            [self.delegate gestureLockView:self didBeginWithPasscode:
            [NSString stringWithFormat:@"%d",touchedButton.tag]];
            //调用 gestureLockView: didBeginWithPasscode: 方法
        }
    }
}
}

```

`touchesMoved:withEvent:` 方法在触摸移动时调用。程序代码如下：

```

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch = [touches anyObject];
    CGPoint locationInContentView = [touch locationInView:self.contentView];
    //判断在 self.contentView.bounds 中是否包含指定的点
    if (CGRectContainsPoint(self.contentView.bounds, locationInContentView)) {
        UIButton *touchedButton = [self _buttonContainsThePoint:
        locationInContentView];
        //判断触摸的按钮或者选择的按钮是否为空
        if (touchedButton != nil && [self.selectedButtons indexOfObject:

```

```

        touchedButton]==NSNotFound) {
            touchedButton.selected = YES;
            [self.selectedButtons addObject:touchedButton]; //添加对象
            //判断 selectedButtons 中的按钮个数是否为 1
            if ([self.selectedButtons count] == 1) {
                if (_delegateFlags.didBeginWithPasscode) {
                    [self.delegate gestureLockView:self didBeginWithPasscode:
                     [NSString stringWithFormat:@"%d",touchedButton.tag]];
                }
            }
        }
        self.trackedLocationInContentView = locationInContentView;
        [self setNeedsDisplay]; //重新绘制
    }
}

```

touchesEnded:withEvent:方法在触摸结束后调用。程序代码如下:

```

- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{
    //判断 selectedButtons 中的按钮个数是否为 1
    if ([self.selectedButtons count] > 0) {
        if (_delegateFlags.didEndWithPasscode) {
            NSMutableArray *passcodeArray = [NSMutableArray array];
            //遍历选中的按钮
            for (UIButton *button in self.selectedButtons) {
                [passcodeArray addObject:[NSString stringWithFormat:@"%d",
                button.tag]]; //添加对象
            }
            [self.delegate gestureLockView:self didEndWithPasscode:[passcode
            Array componentsJoinedByString:@","]];
        }
    }
    //遍历数组
    for (UIButton *button in self.selectedButtons) {
        button.selected = NO;
    }
    [self.selectedButtons removeAllObjects]; //移除选中的按钮
    self.trackedLocationInContentView = CGPointMake(kTrackedLocationInva
    lidInContentView, kTrackedLocationInvalidInContentView);
    [self setNeedsDisplay]; //重新绘制
}

```

(6) 打开 ViewController.h 文件, 编写代码, 实现头文件、遵守协议以及插座变量的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "GestureLockView.h"
@interface ViewController : UIViewController<GestureLockViewDelegate>{
    //插座变量
    IBOutlet GestureLockView *lockView;
    IBOutlet UILabel *label;
}
@end

```

(7) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 9.12 所示。

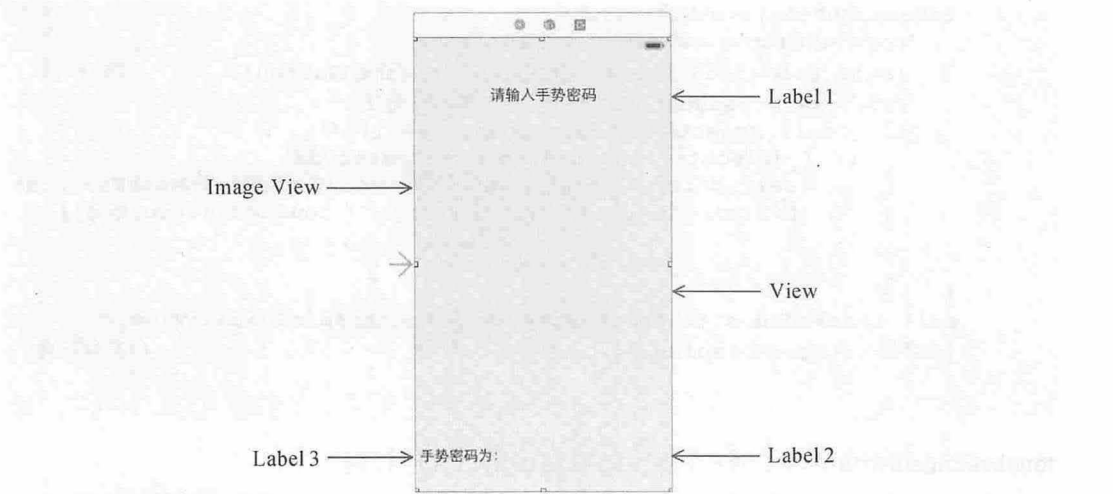


图 9.12 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 9-10 所示。

表 9-10 视图、控件设置

视图、控件	属 性 设 置	其 他
Image View	Image: 3.jpg	与插座变量 containerView 关联
Label1	Text: 请输入手势密码 Font: System 19.0 Alignment: 居中	与插座变量 currentStateLabel 关联
Label2	Text: （空） Alignment: 居中	与插座变量 label 关联
Label3	Text: 手势密码为: Font: System 18.0 Alignment: 居中	
View		Class: GestureLockView 与插座变量 lockView 关联

（8）打开 ViewController.m 文件，编写代码，实现将解锁视图显示在界面上以及解锁密码的显示。程序代码如下：

```
//视图加载后调用，实现初始化
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //对解锁视图进行属性设置
    self.view.backgroundColor = [UIColor whiteColor];
    lockView.normalGestureNodeImage = [UIImage imageNamed:@"1.png"];
    lockView.selectedGestureNodeImage = [UIImage imageNamed:@"2.png"];
    lockView.lineColor = [[UIColor orangeColor] colorWithAlphaComponent:0.3];
    lockView.lineWidth = 12;
```

```

lockView.delegate = self;
lockView.contentInsets = UIEdgeInsetsMake(150, 20, 100, 20);
}
//开始解锁时调用
- (void)gestureLockView:(GestureLockView *)gestureLockView didBeginWith
Passcode:(NSString *)passcode{
}
//解锁结束后调用，并在标签中显示
- (void)gestureLockView:(GestureLockView *)gestureLockView didEndWith
Passcode:(NSString *)passcode{
    label.text=passcode; //设置文本内容
}

```

【代码解析】

由于本实例中的代码和方法非常多，为了方便读者的阅读，笔者绘制了一些执行流程图，如图 9.13 和图 9.14 所示。其中，单击运行按钮一直到在界面显示解锁视图，使用了 `initWithColor:size:`、`_lockViewInitialize`、`initWithCoder:`、`layoutSubviews`、`drawRect:`、`setNormalGestureNodeImage:`、`setSelectedGestureNodeImage:`、`setDelegate:`、`setNumberOfGestureNodes:`、`viewDidLoad` 方法共同实现。它们的执行流程如图 9.13 所示。



图 9.13 程序执行流程

在解锁视图上使用触摸进行解锁使用了 `_buttonContainsThePoint:`、`drawRect:`、`touchesBegan:withEvent:`、`touchesMoved:withEvent:`、`touchesEnded:withEvent:`、`gestureLockView:didBeginWithPasscode:`、`gestureLockView:didEndWithPasscode:` 方法。它们的执行流程如图 9.14 所示。

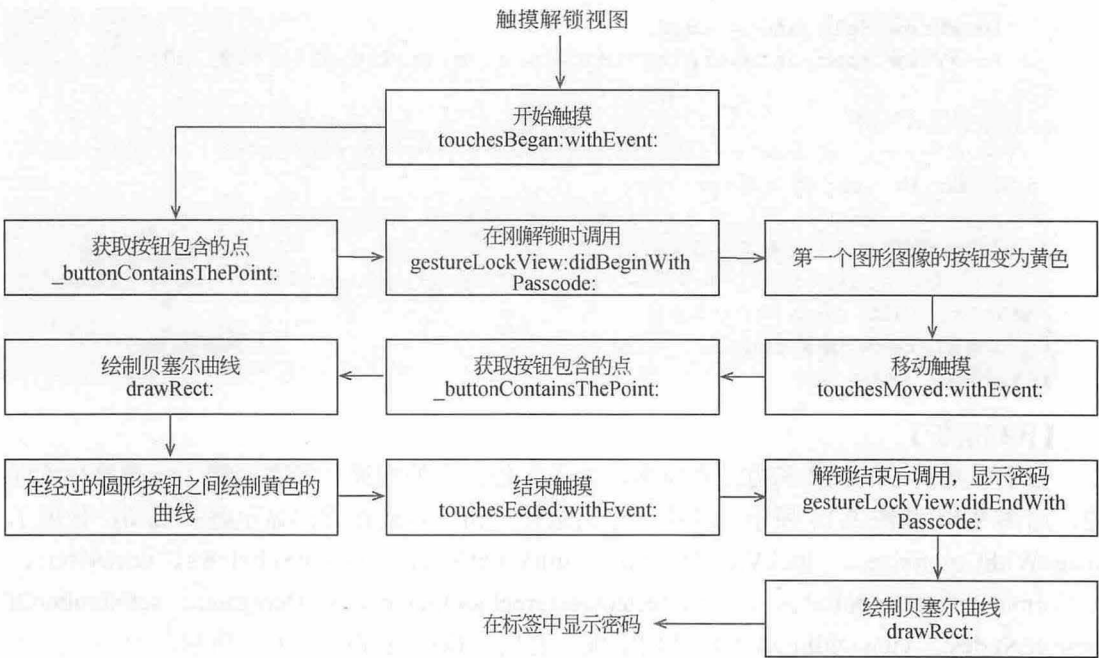


图 9.14 程序执行流程

实例 103 拖动选择图片墙

【实例描述】

本实例的功能是通过拖动的手势去实现图片墙中图片的选取工作。当用户使用拖动手势选择图片后，被选择的图片就会变亮，表示此图片已被选中。运行效果如图 9.15 所示。



图 9.15 运行效果

【实现过程】

(1) 创建一个项目，命名为“拖动选择图片墙”。

(2) 添加图像 0.png~19.png 到创建项目的 Supporting Files 文件夹中。

(3) 创建一个基于 UIView 类的 Positioning 类。

(4) 打开 UIView+Positioning.h 文件, 编写代码, 实现方法的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
@interface UIView (Positioning)
- (void) setOrigin:(float)x :(float)y;           //设置位置
- (void) resetOriginToTopLeft;                   //重新设置位置 (顶部左端)
- (void) resetOriginToTopRight;
- (void) resetOriginToBottomLeft;
@end
```

(5) 打开 UIView+Positioning.m 文件, 编写代码, 实现方法的定义。程序代码如下:

```
//设置位置
- (void) setOrigin:(float)x :(float)y
{
    self.layer.anchorPoint = CGPointMake(x, y);
}
//重新设置位置 (顶部左端)
- (void) resetOriginToTopLeft
{
    [self setOrigin:0 :0];
    [self setCenter:CGPointMake(0, 0)];
}
//重新设置位置 (顶部右端)
- (void) resetOriginToTopRight
{
    [self setOrigin:0 :0];
    [self setCenter:CGPointMake(0, 0)];
}
//重新设置位置 (底部左端)
- (void) resetOriginToBottomLeft
{
    [self setOrigin:self.frame.size.width/2 :self.frame.size.height];
    [self setCenter:CGPointMake(self.frame.size.width/2, self.frame.size.height)];
}
```

(6) 创建一个基于 UICollectionViewController 类的 CollectionViewController 类。

(7) 打开 CollectionViewController.h 文件, 编写代码, 实现头文件、宏定义、实例变量、对象的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import "UIView+Positioning.h"
//宏定义
#define selectedTag 100
#define cellSize 72
.....
#define defaultFontSize 10.0
#define numOfimg 20
@interface CollectionViewController : UICollectionViewController{
    //实例变量
    CGPoint dragStartPt;
    bool dragging;
    //对象
    NSMutableDictionary *selectedIdx;
    NSIndexPath *lastAccessed;
```

```
}
@end
```

(8) 打开 Main.storyboard 文件，删除在画布中的 View Controller 视图控制器。从视图库中拖动 Navigation Controller 导航控制器到创建的项目中。拖动 Collection View Controller 集合视图控制器到画布中。将此视图控制器的 Class 设置为 UICollectionViewController。

(9) 打开 UICollectionViewController.m 文件，编写代码，实现对集合选择器的内容填充以及通过拖动手势选择集合视图控制器中的图像。使用的方法如表 9-11 所示。

表 9-11 UICollectionViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
numberOfSectionsInCollectionView:	获取块数
collectionView:numberOfItemsInSection:	获取展示的 UICollectionViewCell 的个数
collectionView:cellForItemAtIndexPath:	获取 UICollectionView 内容
collectionView:layout:sizeForItemAtIndexPath:	获取每一个定义 UICollectionView 的大小
collectionView:didSelectItemAtIndexPath:	UICollectionView 被选中时调用的方法
collectionView:didDeselectItemAtIndexPath:	UICollectionView 被取消时调用的方法
setCellSelection:selected:	设置 UICollectionViewCell 的背景
resetSelectedCells	重设选择的 UICollectionViewCell
handleGesture:	拖动手势实现选择
selectCellForCollectionView: atIndexPath:	选择 UICollectionViewCell
deselectCellForCollectionView: atIndexPath:	取消 UICollectionViewCell

对集合选择器的内容填充需要使用 viewDidLoad、numberOfSectionsInCollectionView:、collectionView:numberOfItemsInSection:、collectionView:cellForItemAtIndexPath:、collectionView:layout:sizeForItemAtIndexPath:方法实现。这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，viewDidLoad 方法在视图加载后调用，实现初始化功能。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    selectedIdx = [[NSMutableDictionary alloc] init]; //创建可变字典对象
    [self.collectionView registerClass:[UICollectionViewCell class]
    forCellWithReuseIdentifier:@"Cell"];
    [self.collectionView setAllowsMultipleSelection:YES];
    //为导航栏添加按钮
    UIBarButtonItem *btnReset = [[UIBarButtonItem alloc] initWithBarButton
    SystemItem:UIBarButtonSystemItemRefresh target:self action:@selector
    (resetSelectedCells)];
    self.navigationItem.rightBarButtonItem = btnReset;//设置栏右边的按钮条目
    //创建并设置拖动手势
    UIPanGestureRecognizer *gestureRecognizer = [[UIPanGestureRecognizer
    alloc] initWithTarget:self action:@selector(handleGesture:)];
    [self.view addGestureRecognizer:gestureRecognizer];//添加手势识别器对象
    [gestureRecognizer setMinimumNumberOfTouches:1];
    [gestureRecognizer setMaximumNumberOfTouches:1];
}
```

collectionView:cellForItemAtIndexPath:方法实现对 UICollectionView 内容进行获取。程

序代码如下:

```
- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
  cellForItemAtIndexPath:(NSIndexPath *)indexPath{
    static NSString *identifier = @"Cell";
    UICollectionViewCell *cell = (UICollectionViewCell *)[collectionView
    dequeueReusableCellWithReuseIdentifier:identifier forIndexPath:indexPath];
    //判断是否被选择
    if (![cell viewWithTag:selectedTag])
    {
        UILabel *selected = [[UILabel alloc] initWithFrame:CGRectMake(0,
        cellSize - textLabelHeight, cellSize, textLabelHeight)];
        selected.backgroundColor = [UIColor darkGrayColor]; //设置背景颜色
        selected.textColor = [UIColor whiteColor];           //设置文本颜色
        selected.text = @"SELECTED";
        selected.textAlignment = NSTextAlignmentCenter;     //设置对齐方式
        selected.font = [UIFont systemFontOfSize:defaultFontSize];
        selected.tag = selectedTag;
        selected.alpha = cellAHidden;                         //设置透明度
        [cell.contentView addSubview:selected];
    }
    cell.backgroundView = [[UIImageView alloc] initWithImage:[UIImage
    imageNamed:[NSString stringWithFormat:@"%ld.png", [indexPath row] %
    numOfimg]]];
    [[cell viewWithTag:selectedTag] setAlpha:cellAHidden];
    cell.backgroundView.alpha = cellADeactive;               //设置透明度
    bool cellSelected = [selectedIdx objectForKey:[NSString stringWithFormat:
    @"%ld", (long)indexPath.row]];
    [self setCellSelection:cell selected:cellSelected]; //设置选中的单元格
    return cell;
}
```

通过拖动手势选择集合视图控制器中的图像, 需要使用 collectionView:didSelectItemAtIndexPath:、collectionView:didDeselectItemAtIndexPath:、setCellSelection:selected:、resetSelectedCells、handleGesture:、selectCellForCollectionView:atIndexPath:、deselectCellForCollectionView:atIndexPath:方法。其中, handleGesture:方法实现使用拖动手势选择图像的功能。程序代码如下:

```
- (void) handleGesture:(UIPanGestureRecognizer *)gestureRecognizer
{
    //获取触摸位置的 x 与 y 的位置
    float pointerX = [gestureRecognizer locationInView:self.collectionView].x;
    float pointerY = [gestureRecognizer locationInView:self.collectionView].y;
    //遍历
    for (UICollectionViewCell *cell in self.collectionView.visibleCells) {
        float cellSX = cell.frame.origin.x; //获取单元格的 x 值
        float cellEX = cell.frame.origin.x + cell.frame.size.width;
        float cellSY = cell.frame.origin.y; //获取单元格的 y 值
        float cellEY = cell.frame.origin.y + cell.frame.size.height;
        //判断触摸的位置
        if (pointerX >= cellSX && pointerX <= cellEX && pointerY >= cellSY
        && pointerY <= cellEY)
        {
            //当前触摸的 UICollectionViewCell
            NSIndexPath *touchOver = [self.collectionView indexPathForCell:cell];
            //判断当前触摸的是否为上次触摸的
            if (lastAccessed != touchOver)

```



```

    {
        //判断 UICollectionViewCell 是否被选择
        if (cell.selected) {
            [self deselectCellForCollectionView:self.collectionView
             atIndexPath:touchOver];
        }
        else {
            [self selectCellForCollectionView:self.collectionView
             atIndexPath:touchOver];
        }
        lastAccessed = touchOver;
    }
}
//判断手势的状态
if (gestureRecognizer.state == UIGestureRecognizerStateEnded)
{
    lastAccessed = nil;
    self.collectionView.scrollEnabled = YES;
}
}

```

【代码解析】

本实例关键功能是图像的选择。下面就是这个知识点的详细讲解。

在本实例中，选择图像，首先需要判断当前触摸的位置是否在某一个 UICollectionViewCell 的矩形区域，代码如下。

```

float cellSX = cell.frame.origin.x;
float cellEX = cell.frame.origin.x + cell.frame.size.width;
float cellSY = cell.frame.origin.y;
float cellEY = cell.frame.origin.y + cell.frame.size.height;
if (pointerX >= cellSX && pointerX <= cellEX && pointerY >= cellSY && pointerY <= cellEY)
{
    .....
}

```

如果当前触摸的位置在某一个 UICollectionViewCell 的矩形区域内，再次判断当前触摸的是否和上一次触摸的一样，如果不一样，再进行判断 UICollectionViewCell 是否被选择：如果选择，就调用 `deselectCellForCollectionView: atIndexPath:` 取消选择；如果没有选择，就调用 `selectCellForCollectionView: atIndexPath:` 进行选择。这就是图像的选择过程。代码如下：

```

NSIndexPath *touchOver = [self.collectionView indexPathForCell:cell];
if (lastAccessed != touchOver)
{
    if (cell.selected)
        [self deselectCellForCollectionView:self.collectionView atIndexPath:
         touchOver];
    else
        [self selectCellForCollectionView:self.collectionView atIndexPath:
         touchOver];
}

```

第 10 章 照片库与相机

在生活中人们常常会使用相机功能将自己的点点滴滴记录下，作为美好的回忆。所以手机中的相机功能就成为了必不可少的软件应用。在本章中将为读者实现一些关于照片库以及相机的小应用。

实例 104 更改应用程序的背景

【实例描述】

在本实例实现的功能是通过图像选取控制器的 3 种不同的图像来源来选取照片，进而实现更换应用程序背景的功能。当用户单击“选择图像”按钮后，弹出具有 4 个按钮的警告视图；当用户选择其中一个按钮后，就会实现对应的操作，例如，当用户选择“照片库”按钮后，就会打开照片库的根目录，进而实现照片的选取功能，当选取完成后，就会退出照片库，将选取的图像显示在应用程序的背景中。运行效果如图 10.1 所示。



图 10.1 运行效果

【实现过程】

(1) 创建一个项目，命名为“更改应用程序的背景”。

(2) 打开 ViewController.h 文件，编写代码，实现遵守协议、对象、插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface
ViewController :UIViewController<UIImagePickerControllerDelegate,
UINavigationControllerDelegate>{
    UIImagePickerController *imagePicker;
    IBOutlet UIImageView *imageView;           //插座变量
}
- (IBAction) show:(id) sender;
@end
```

(3) 打开 Main.storyboard 文件，拖动 Navigation Controller 导航控制器到画布中，将此控制器关联的根视图设置为 View Controller 视图控制器。对 View Controller 视图控制器的设计界面进行设计，效果如图 10.2 所示。

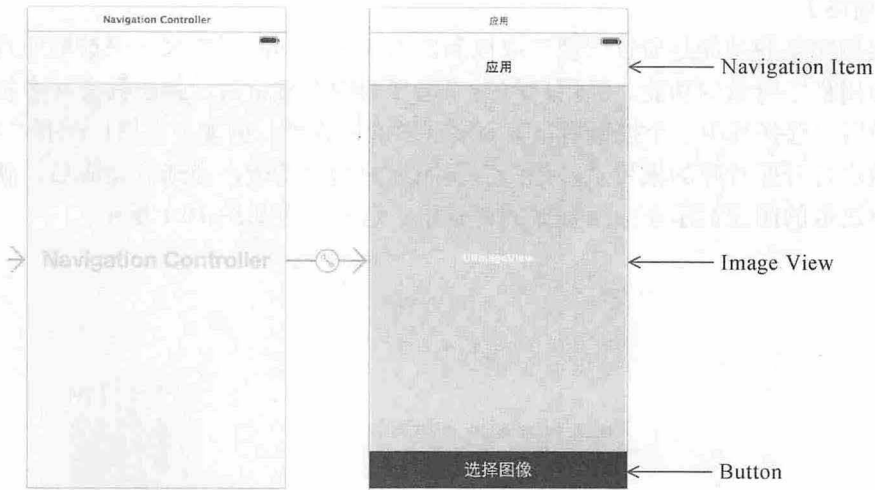


图 10.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 10-1 所示。

表 10-1 视图、控件设置

视图、控件	属性设置	其他
Navigation Item	Title: 应用	
Image View		与插座变量 imageView 关联
Button	Title: 选择图像 Font: System 21.0 Text Color: 白色 Background: 黑色	与动作 show:关联

(4) 打开 ViewController.m 文件，编写代码，实现单击“选择图像”按钮进行图像的选择。使用的方法如表 10-2 所示。

表 10-2 ViewController.m文件中方法总结

方 法	功 能
show:	单击按钮，弹出警告视图
alertView:clickedButtonAtIndex:	实现警告视图的响应
imagePickerController:didFinishPickingMediaWithInfo:	用户选择照片后调用，实现将照片显示在指定的图像视图中

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，alertView:clickedButtonAtIndex:方法实现对警告视图中按钮选择后的响应。程序代码如下：

```
- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)
buttonIndex{
    NSString *str=[alertView buttonTitleAtIndex:buttonIndex];
    if ([str isEqualToString:@"照片库"]) {
        //打开照片库
        imagePicker=[[UIImagePickerController alloc]init];
        imagePicker.sourceType=UIImagePickerControllerSourceTypePhotoLibrary;
                                                    //设置照片来源
        imagePicker.delegate=self;                    //设置委托
        imagePicker.allowsEditing=YES;                //可以编辑
        [self presentViewController:imagePicker animated:YES completion:
        NULL];                                          //显示
    }else if ([str isEqualToString:@"相册"]) {
        //打开相册
        imagePicker=[[UIImagePickerController alloc]init];
        //设置照片来源
        imagePicker.sourceType=UIImagePickerControllerSourceTypeSavedPhotosAlbum;
        imagePicker.delegate=self;                    //设置委托
        imagePicker.allowsEditing=YES;                //可以编辑
        [self presentViewController:imagePicker animated:YES completion:NULL];
    }else{
        if ( [UIImagePickerController isSourceTypeAvailable:UIImagePicker
        ControllerSourceTypeCamera]) {
            //打开保存相机中图像的相册
            imagePicker=[[UIImagePickerController alloc]init];
            //设置照片来源
            imagePicker.sourceType=UIImagePickerControllerSourceTypeSaved
            PhotosAlbum;
            imagePicker.delegate=self;                    //设置委托
            imagePicker.allowsEditing=YES;                //可以编辑
            [self presentViewController:imagePicker animated:YES completion:
            NULL];
        }else{
            UIAlertView *alert=[[UIAlertView alloc]initWithTitle:@"提示"
            message:@"设备不支持自定义" delegate:nil cancelButtonTitle:
            @"Cancel" otherButtonTitles: nil];            //创建警告视图
            [alert show];
        }
    }
}
```

imagePickerController:didFinishPickingMediaWithInfo:方法在用户选择图片后调用，实现将选择的图像设置为 imageView 的图像。程序代码如下：

```

-(void)imagePickerController:(UIImagePickerController *)picker didFinish
PickingMediaWithInfo:(NSDictionary *)info{
    [imagePicker dismissViewControllerAnimated:YES completion:NO];
    UIImage *im=[info objectForKey:UIImagePickerControllerEditedImage];
    imageView.image=im; //设置图像
}

```

【代码解析】

本实例关键功能是照片库、相册、照相机的打开，以及照相机设备的判断。下面就是这个知识点的详细讲解。

1. 照片库、相册、照相机的打开

UIImagePickerController 类的 sourceType 属性表示图像选取控制器的 3 种不同的媒体来源模式。其语法形式如下：

```
@property (nonatomic) UIImagePickerControllerSourceType sourceType;
```

其中，3 种来源模式如下所示：

- ❑ UIImagePickerControllerSourceTypePhotoLibrary：表示照片库模式。图像选取控制器以该模式显示时会浏览系统照片库的根目录。
- ❑ UIImagePickerControllerSourceTypeSavedPhotosAlbum：相册模式。图像选取控制器以该模式显示时会浏览相册目录。
- ❑ UIImagePickerControllerSourceTypeCamera：相机模式。图像选取控制器以该模式显示时可以进行拍照或摄像。

在本实例中就是使用了此方法实现了照片库、相册、照相机的打开。以打开照片库为例，程序代码如下：

```
imagePicker.sourceType=UIImagePickerControllerSourceTypePhotoLibrary;
```

2. 照相机设备的判断

在使用照相机之前，务必对照相机设备进行判断，检查该设备中是否是可用照相机，从而可以减少不必要的错误。例如，在模拟器中就不可以使用照相机设备，从而就不会打开照相机。对照相机是否可用的判断，需要使用 UIImagePickerController 的 isSourceTypeAvailable:方法，其语法形式如下：

```
+ (BOOL)isSourceTypeAvailable:(UIImagePickerControllerSourceType) sourceType;
```

其中，(UIImagePickerControllerSourceType)sourceType 表示 3 种不同的媒体来源模式。在本实例中就使用了 isSourceTypeAvailable:方法实现了对照相机是否可用的判断。程序代码如下：

```

if ( [UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
    .....
}else{
    .....
}

```

实例 105 自定义相机

【实例描述】

在 iOS 中,有很多拍照功能的应用程序都不是使用 iOS 自带的拍照界面,而是各种很漂亮、个性的自定义相机。在本实例中就为各位读者也实现一个自定义的相机,让拍照的应用程序变得不再单一。运行效果如图 10.3 所示。



图 10.3 运行效果

【实现过程】

- (1) 创建一个项目,命名为“自定义相机”。
- (2) 添加图像 1.png 和 2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件,编写代码,实现宏定义、遵守协议、插座变量、对象、属性、动作、方法的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
//宏定义
#define GET_IMAGE(__NAME__, __TYPE__) [UIImage imageNamed:[NSBundle mainBundle] pathForResource:__NAME__ ofType:__TYPE__]
@interface ViewController : UIViewController<UIImagePickerControllerDelegate, UINavigationControllerDelegate>{
    IBOutlet UIScrollView *scrollView; //插座变量
    NSMutableArray *imageArray;
}
@property (nonatomic, retain) UIImagePickerController *imagePickerController; //属性
//方法
- (IBAction)show:(id)sender;
- (void)setupImagePicker:(UIImagePickerControllerSourceType)sourceType;
@end
```

- (4) 打开 Main.storyboard 文件,对 View Controller 视图控制器的设计界面进行设计,效果如图 10.4 所示。

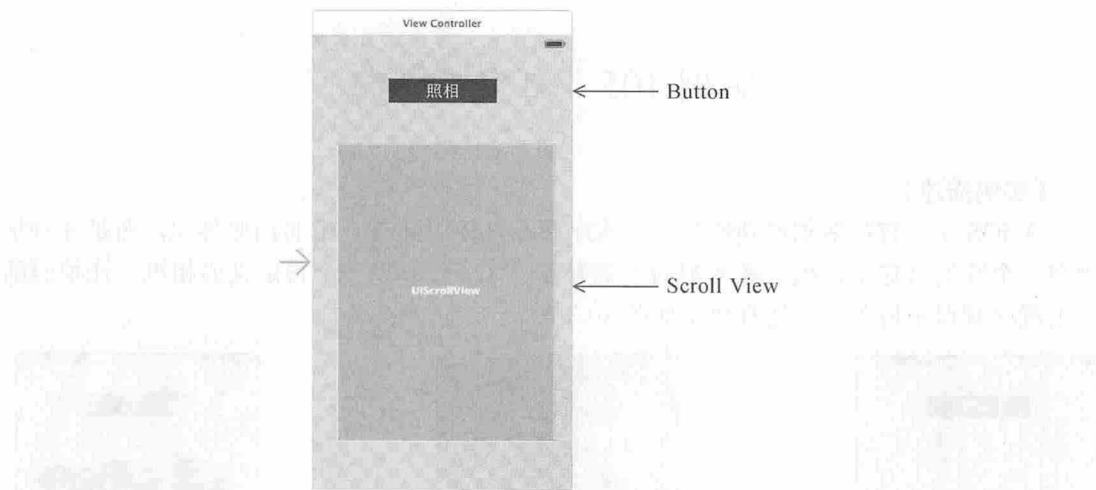


图 10.4 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 10-3 所示。

表 10-3 视图、控件设置

视图、控件	属 性 设 置	其 他
设计界面	Background: 浅灰色	
Scroll View		与插座变量 scrollView 关联
Button	Title: 照相 Font: System 20.0 Text Color: 白色 Background: 黑色	与动作 show:关联

（5）打开 ViewController.m 文件，编写代码，实现自定义相机的功能。其中包含 3 个部分：相机界面的设计以及相机的拍照以及照片的保存功能。使用的方法如表 10-4 所示。

表 10-4 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
show:	单击“相机”按钮，弹出相机应用
setupImagePicker:	设计相机的界面
changeCameraDevice:	改变相机的摄像头
stillImage:	拍照
doneAction	改变相机应用
refreshImage	刷新照片
imagePickerControllerDidCancel:	关闭相机应用
imagePickerController:didFinishPickingMediaWithInfo:	用户选择照片后调用，实现将照片显示在滚动视图中

相机界面的设计需要使用 viewDidLoad、setupImagePicker:方法。这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，setupImagePicker:方法实现为相机中的界面添加必需的功能按钮。程序代码如下：


```

- (void)setupImagePicker:(UIImagePickerControllerSourceType)sourceType
{
    self.imagePickerController.sourceType = sourceType;    //设置照片来源
    //盘发照片来源是否为 UIImagePickerControllerSourceTypeCamera
    if (sourceType == UIImagePickerControllerSourceTypeCamera)
    {
        // 不使用系统的控制界面
        self.imagePickerController.showsCameraControls = NO;
        UIToolbar *controlView = [[UIToolbar alloc] initWithFrame:CGRectMake(
            0, self.view.frame.size.height-88, self.view.frame.size.width, 60)];
        //创建工具栏
        controlView.autoresizingMask = UIViewAutoresizingFlexibleTopMargin
        | UIViewAutoresizingFlexibleWidth;    //设置自适应
        //拍照
        UIButton *cameraBtn = [UIButton buttonWithType:UIButtonTypeCustom];
        cameraBtn.frame = CGRectMake(0, 0, 60, 60);    //设置框架
        cameraBtn.showsTouchWhenHighlighted = YES;
        [cameraBtn setImage:[UIImage imageNamed:@"1.png"], forState:UIControlStateNormal]; //设置图像
        [cameraBtn addTarget:self action:@selector(stillImage:) forControlEvents:
            UIControlEventTouchUpInside];    //添加动作
        UIBarButtonItem *takePicItem = [[UIBarButtonItem alloc] initWith
            CustomView:cameraBtn];
        //摄像头切换
        UIButton *cameraDevice = [UIButton buttonWithType:UIButtonTypeCustom];
        cameraDevice.frame = CGRectMake(0, 0, 40, 40);    //设置框架
        cameraDevice.showsTouchWhenHighlighted = YES;
        [cameraDevice setImage:[UIImage imageNamed:@"2.png"], forState:UIControlStateNormal];
        [cameraDevice addTarget:self action:@selector(changeCameraDevice:)
            forControlEvents:UIControlEventTouchUpInside];    //添加动作
        UIBarButtonItem *cameraDeviceItem = [[UIBarButtonItem alloc] initWith
            CustomView:cameraDevice];
        //判断是否支持前置摄像头
        if (![UIImagePickerController isCameraDeviceAvailable:UIImage
            PickerControllerCameraDeviceFront]) {
            cameraDeviceItem.enabled = NO;
        }
        //取消、完成
        UIBarButtonItem *doneItem = [[UIBarButtonItem alloc] initWithTitle:
            @"取消" style:UIBarButtonItemStyleBordered target:self action:
            @selector(doneAction)];    //创建取消按钮
        //创建完成按钮
        UIBarButtonItem *spItem = [[UIBarButtonItem alloc] initWithBarButton
            SystemItem:UIBarButtonItemSystemItemFlexibleSpace target:nil action:nil];
        NSArray *items = [NSArray arrayWithObjects:cameraDeviceItem,
            spItem,takePicItem,spItem,doneItem, nil];
        [controlView setItems:items];    //设置条目
        self.imagePickerController.cameraOverlayView = controlView;
        controlView = nil;
    }
}

```

相机拍照功能的实现需要使用 show:、changeCameraDevice:、stillImage:、doneAction 方法实现。其中, show:方法实现单击“相机”按钮的触发,弹出自定义相机。程序代码如下:

```

- (IBAction)show:(id)sender {
    UIImagePickerControllerSourceType sourceType = UIImagePickerControllerSourceTypeCamera;
    //判断是否存在相机
    if (![UIImagePickerController isSourceTypeAvailable: UIImagePickerControllerSourceTypeCamera]) {
        sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    }
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    picker.delegate = self; //设置委托
    picker.allowsEditing = NO; //不可编辑
    self.imagePickerController = picker;
    [self setupImagePicker:sourceType];
    picker = nil;
    [self presentViewController:self.imagePickerController animated:YES completion:nil]; //显示
}

```

changeCameraDevice:方法实现改变相机的摄像头。程序代码如下:

```

- (void)changeCameraDevice:(id)sender
{
    //判断设备是否为后置摄像头
    if (self.imagePickerController.cameraDevice == UIImagePickerControllerCameraDeviceRear) {
        //判断是否支持前置摄像头
        if ([UIImagePickerController isCameraDeviceAvailable:UIImagePickerControllerCameraDeviceFront]) {
            self.imagePickerController.cameraDevice = UIImagePickerControllerCameraDeviceFront;
        }
    }
    else {
        self.imagePickerController.cameraDevice = UIImagePickerControllerCameraDeviceRear;
    }
}

```

stillImage:方法实现拍照功能。程序代码如下:

```

- (void)stillImage:(id)sender
{
    [self.imagePickerController takePicture];
}

```

照片的保存需要使用 refreshImage、imagePickerControllerDidCancel、imagePickerController:didFinishPickingMediaWithInfo:方法。其中, refreshImage 方法实现刷新照片的功能。程序代码如下:

```

- (void)refreshImage
{
    //移除所有旧的子 view
    for (UIView *subView in scrollView.subviews) {
        [subView removeFromSuperview];
    }
    scrollView.contentSize = CGSizeMake(280*imageArray.count, 320);
}

```

```

//设置可滚动区域的尺寸
//遍历数组
for (int i = 0; i < imageArray.count; i++) {
    UIImageView *imageView = [[UIImageView alloc] initWithImage:
        [imageArray objectAtIndex:i]];
    imageView.contentMode = UIViewContentModeScaleAspectFit;
    //设置图像的显示模式
    imageView.frame = CGRectMake(i*280, 0, 280, 320); //设置框架
    [scrollView addSubview:imageView]; //添加视图对象
}
//清空数组
[imageArray removeAllObjects]; //移除所有的对象
[scrollView setContentOffset:CGPointZero];
}

```

imagePickerController:didFinishPickingMediaWithInfo:方法是在用户选择照片后调用, 实现将照片显示在滚动视图中。程序代码如下:

```

- (void)imagePickerController:(UIImagePickerController *)picker didFinish
PickingMediaWithInfo:(NSDictionary *)info
{
    [imageArray addObject:[info objectForKey:UIImagePickerController
OriginalImage]]; //添加对象
    [picker dismissViewControllerAnimated:YES completion:nil];
    [self refreshImage];
}

```

【代码解析】

本实例关键功能是自定义相机的界面以及相机的拍照功能。下面依次讲解这两个知识点。

1. 自定义相机的界面

UIImagePickerController 的 cameraOverlayView 属性实现在选取控制器之上的图像自定义视图。注意, 只有当源是 UIImagePickerControllerSourceTypeCamera 时才有效。其语法形式如下:

```
@property (nonatomic, retain) UIView *cameraOverlayView;
```

在本实例中就是使用了 cameraOverlayView 属性改变相机的界面, 代码如下:

```
self.imagePickerController.cameraOverlayView = controlView;
```

2. 相机的拍照

UIImagePickerController 的 takePicture 方法实现了使用相机捕获图像的功能, 其语法形式如下:

```
- (void)takePicture;
```

在本实例中就使用 takePicture 方法实现了单击拍照按钮后进行拍照的功能, 代码如下:

```
[self.imagePickerController takePicture];
```

实例 106 狙 击 枪

【实例描述】

在现在的 iOS 应用中增强现实的例子数不胜数。本实例就为各位读者也实现一个增强现实感的实例——狙击枪。当用户单击“狙击枪”按钮后，就会弹出自定义的相机，当用户单击 FIRE!按钮后，就会发出子弹，当到达指定的位置时，子弹就会变成血。运行效果如图 10.5 所示。

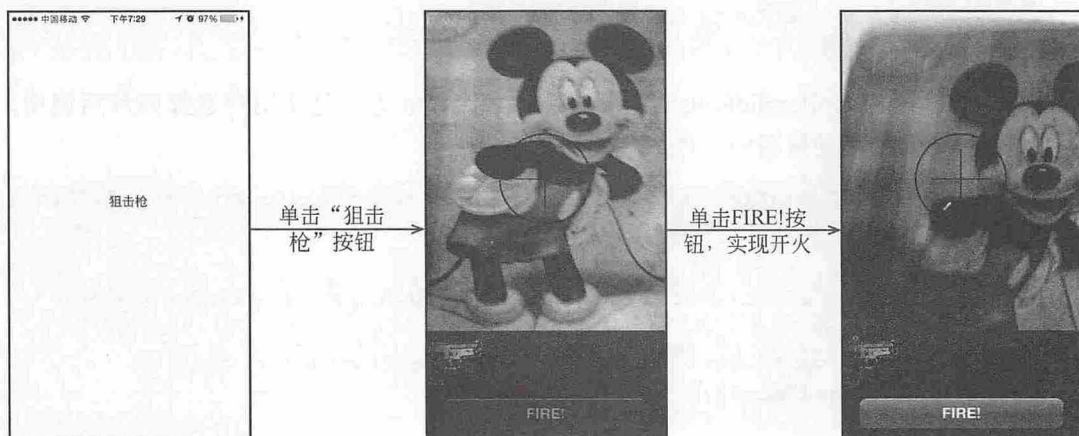


图 10.5 运行效果

【实现过程】

- (1) 创建一个项目，命名为“狙击枪”。
- (2) 添加音频文件 1.caf 到创建项目的 Supporting Files 文件夹中。
- (3) 添加图像 2.png、3.png、4.png、5.png 到创建项目的 Supporting Files 文件夹中。
- (4) 添加 AVFoundation.framework 到创建的项目中。
- (5) 创建一个基于 UIView 类的 View 类。
- (6) 打开 View.h 文件，编写代码，实现头文件、宏定义以及对象的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
//宏定义
#define BOTTOM_BAR_HEIGHT 60
#define VIEW_FINDER_RADIUS 60
#define VIEW_FIRE_AREA_RADIUS 30
#define CGRectMidPoint(rect) CGPointMake(CGRectGetMidX(rect), CGRectGetMidY(rect))
@interface View : UIView
{
    AVAudioPlayer *player;
}
@end
```


(7) 打开 View.m 文件, 编写代码, 实现狙击枪的视图设置。使用的方法如表 10-5 所示。

表 10-5 View.m 文件中方法总结

方 法	功 能
initWithFrame:	初始化
soundNamed:	获取声音
drawViewFinder	绘制靶心
drawBottomBar	绘制底栏
drawGun	绘制枪
drawRect:	绘制视图
showSplashAnimationAtPoint:	显示血飞溅的动画
fire	单击按钮后, 发射子弹

这里需要讲解几个重要的方法 (其他方法请读者参考源代码)。其中, soundNamed: 方法实现对声音的获取。

```

- (AVAudioPlayer *)soundNamed:(NSString *)name {
    NSString * path;
    AVAudioPlayer *sound;
    NSError * err;
    path = [[[NSBundle mainBundle] resourcePath] stringByAppending
   PathComponent:name];
    //判断文件是否存在
    if ([[NSFileManager defaultManager] fileExistsAtPath:path]) {
        NSURL *url = [NSURL fileURLWithPath:path];
        sound = [[AVAudioPlayer alloc] initWithContentsOfURL:url error:&err];
        //判断声音是否存在
        if (!sound) {
            //声音不存在
            NSLog(@"Sound named '%@' had error %@", name, [err localized
            Description]);
        } else {
            //声音存在
            [sound prepareToPlay];
        }
    } else {
        NSLog(@"Sound file '%@' doesn't exist at '%@'", name, path);
    }
    return sound;
}

```

drawViewFinder 方法实现对靶心的绘制。程序代码如下:

```

- (void)drawViewFinder
{
    UIBezierPath *bezierPath = [UIBezierPath bezierPath]; //创建贝塞尔路径
    CGPoint center = CGPointMake(CGRectGetMidX(self.frame), CGRectGetMidY
    (self.frame) - BOTTOM_BAR_HEIGHT);
    //绘制弧
    [bezierPath addArcWithCenter:center radius:VIEW_FINDER_RADIUS startAngle:0
    endAngle:360 clockwise:YES];
}

```



```

//绘制线段
[bezierPath moveToPoint:CGPointMake(center.x - VIEW_FINDER_RADIUS + 20,
center.y)];
//设置开始点
[bezierPath addLineToPoint:CGPointMake(center.x + VIEW_FINDER_RADIUS -
20, center.y)];
[bezierPath moveToPoint:CGPointMake(center.x, center.y - VIEW_FINDER_
RADIUS + 20)];
[bezierPath addLineToPoint:CGPointMake(center.x, center.y + VIEW_
FINDER_RADIUS - 20)];//设置结束点
[bezierPath setLineWidth:2];
//设置线宽
[[UIColor blackColor] setStroke];
[bezierPath stroke];
}

```

drawBottomBar 方法实现底边栏的绘制。程序代码如下：

```

- (void)drawBottomBar
{
    CGContextRef context = UIGraphicsGetCurrentContext(); //创建图形上下文
    CGRect bottomBar = CGRectMake(0, self.frame.size.height - BOTTOM_
BAR_HEIGHT, self.frame.size.width, BOTTOM_BAR_HEIGHT);
    [[UIColor blackColor] setFill];
    CGContextFillRect(context, bottomBar);
    //绘制
    //创建并设置按钮对象
    UIButton *fireButton = [UIButton buttonWithType:UIButtonTypeCustom];
    //添加动作
    [fireButton addTarget:self action:@selector(fire) forControlEvents:
UIControlEventTouchUpInside];
    float buttonWidth = 279;
    float buttonHeight = 48;
    fireButton.frame = CGRectMake(self.frame.size.width / 2 - buttonWidth
/ 2, self.frame.size.height - BOTTOM_BAR_HEIGHT + 6, buttonWidth,
buttonHeight);
    //设置框架
    [fireButton setImage:[UIImage imageNamed:@"2.png"] forState:UIControlStateNormal];
    [self addSubview:fireButton];
}

```

showSplashAnimationAtPoint:方法实现血花飞溅的动画效果。程序代码如下：

```

- (void)showSplashAnimationAtPoint:(CGPoint)aPoint
{
    UIImage *splashImage = [UIImage imageNamed:@"5.png"];
    UIImageView *splash = [[UIImageView alloc] initWithFrame:CGRectMake
(aPoint.x, aPoint.y, splashImage.size.width, splashImage.size.height)];
    splash.image = splashImage;
    //设置图像
    [self addSubview:splash];
    //动画
    [UIView animateWithDuration:1 animations:^(
        [splash setAlpha:0];
        //设置透明度
    ) completion:^(BOOL finished) {
        [splash removeFromSuperview];
        //移除指定视图对象
    }];
}

```

fire 方法实现单击底边栏发出子弹的效果。程序代码如下：

```
- (void)fire
{
    [player play];
    //创建并设置图像视图，用来显示子弹
    UIImage *bulletImage = [UIImage imageNamed:@"4.png"];
    UIImageView *bullet = [[UIImageView alloc] initWithFrame:CGRectMake(60,
self.frame.size.height - 140, bulletImage.size.width, bulletImage.size.
height)];
    bullet.image = bulletImage;                                //设置图像
    [self addSubview:bullet];
    //动画
    [UIView animateWithDuration:0.5 animations:^(
        float xDelta = 8;
        float yDelta = 8;
        float x = (arc4random() % VIEW_FIRE_AREA_RADIUS) + CGRectGetMidX
        (self.frame) - (VIEW_FIRE_AREA_RADIUS / 2) - xDelta;
                                                //生成随机数赋值给 x
        float y = (arc4random() % VIEW_FIRE_AREA_RADIUS) + CGRectGetMidY (self.frame)
        - BOTTOM_BAR_HEIGHT - (VIEW_FIRE_AREA_RADIUS / 2) - yDelta;
        //设置框架
        bullet.frame = CGRectMake(x, y, bulletImage.size.width / 2,
        bulletImage.size.height / 2);
    ) completion:^(BOOL finished) {
        [self showSplashAnimationAtPoint:CGPointMake(bullet.frame.origin.x,
        bullet.frame.origin.y)];
        [bullet removeFromSuperview];                                //移除指定的视图
    }];
}
```

(8) 打开 ViewController.h 文件，编写代码，实现头文件、对象以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "View.h"
@interface ViewController : UIViewController{
    View *overlayView;
    UIImagePickerController *imagePickerController;
}
- (IBAction)show:(id)sender;
@end
```

(9) 打开 Main.storyboard 文件，从视图库中拖动 Button 按钮控件到设计界面中，将此控件的 Title 属性设置为“狙击枪”；将 Font 属性设置为 System 18.0；将此控件与 show: 方法关联。

(10) 打开 ViewController.m 文件，编写代码，实现自定义相机的弹出。程序代码如下：

```
- (IBAction)show:(id)sender {
    imagePickerController = [[UIImagePickerController alloc] init];
    imagePickerController.sourceType = UIImagePickerControllerSourceTypeCamera; //设置照片来源
    [self presentViewController:imagePickerController animated:YES completion:nil];
    overlayView = [[View alloc] initWithFrame:imagePickerController.view.frame];
}
```

```
overlayView.backgroundColor = [UIColor clearColor]; //设置背景颜色
imagePickerController.cameraOverlayView = overlayView;
imagePickerController.showsCameraControls = NO;
}
```

【代码解析】

本实例关键功能是靶心、手枪等的显示以及单击 FIRE!按钮发射子弹。下面依次讲解这两个知识点。

1. 靶心、手枪等的显示

在本实例中靶心、手枪、子弹等的显示都是通过使用 UIImagePickerController 的 cameraOverlayView 属性实现的，它的功能是实现在选取控制器之上的图像自定义视图（注意只有当源是 UIImagePickerControllerSourceTypeCamera 时有效）。代码如下：

```
imagePickerController.cameraOverlayView = overlayView;
```

其中，overlayView 表示绘制有靶心、手枪、子弹等视图的对象。

2. 单击 FIRE!按钮发射子弹

在本实例中单击 FIRE!按钮后发射子弹以及子弹到达某一位置后变为血的动画效果都是通过 UIView 的 animateWithDuration:animations:completion:方法实现的。程序代码如下：

```
[UIView animateWithDuration:0.5 animations:^(
    //发射子弹
    float xDelta = 8;
    float yDelta = 8;
    float x = (arc4random() % VIEW_FIRE_AREA_RADIUS) + CGRectGetMidX
    (self.frame) - (VIEW_FIRE_AREA_RADIUS / 2) - xDelta;
    float y = (arc4random() % VIEW_FIRE_AREA_RADIUS) + CGRectGetMidY(self.
    frame) - BOTTOM_BAR_HEIGHT - (VIEW_FIRE_AREA_RADIUS / 2) - yDelta;
    bullet.frame = CGRectMake(x, y, bulletImage.size.width / 2,
    bulletImage.size.height / 2);
} completion:^(BOOL finished) {
    //变为血
    [self showSplashAnimationAtPoint:CGPointMake(bullet.frame.origin.x,
    bullet.frame.origin.y)];
    [bullet removeFromSuperview];
}];
```

实例 107 水印相机

【实例描述】

水印相机是 QQ 空间 3.5 版本的应用。水印相机可以在用户分享的照片基础上加印上地理位置、天气、PM2.5 情况、当前时间，甚至周围声音分贝数等信息。在本实例中就为各位读者实现此应用。当用户单击“相机”按钮，就会弹出具有水印的照相。当用户单击拍照按钮，拍照的水印照片就会显示在滚动视图中。当单击 Save 按钮，就会将照片保存在相册中。运行效果如图 10.6 所示。



图 10.6 运行效果

【实现过程】

- (1) 创建一个项目，命名为“水印相机”。
- (2) 添加图像 1.png、2.png 到创建项目的 Supporting Files 文件夹中。
- (3) 打开 ViewController.h 文件，编写代码，实现宏定义、遵守协议、对象、插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#define GET_IMAGE(__NAME__, __TYPE__) [UIImage imageWithContentsOfFile:
[[NSBundle mainBundle] pathForResource:__NAME__ ofType:__TYPE__]]
@interface ViewController : UIViewController<UINavigationControllerDelegate,
UIImagePickerControllerDelegate>{
    //对象
    UIImagePickerController *imagePickerController;
    NSMutableArray *imageArray;
    IBOutlet UIScrollView *scrollView;           //滚动视图的插座变量
}
//动作
- (IBAction)show:(id)sender;
- (IBAction)save:(id)sender;
@end
```

- (4) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 10.7 所示。

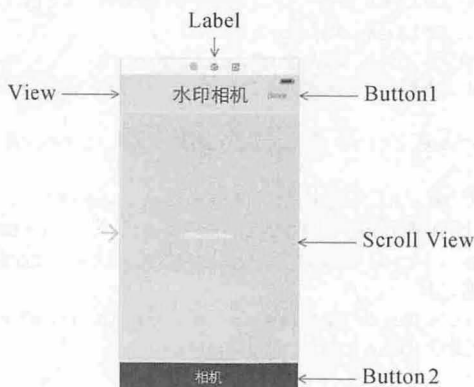


图 10.7 运行效果

需要添加的视图、控件以及对它们的设置如表 10-6 所示。

表 10-6 视图、控件设置

视图、控件	属 性 设 置	其 他
View	Background: 浅灰色	
Button1	Title: Save	与动作 save:关联
Label	Text: 水印相机 Font: System 34.0 Alignment: 居中	
Scroll View		与插座变量 scrollView 关联
Button2	Title: 相机 Font: System 25.0 Text Color: 白色 Background: 黑色	与动作 show:关联

(5) 打开 ViewController.m 文件, 编写代码, 实现对相机的设置、照相以及保存照片的功能。使用的方法如表 10-7 所示。

表 10-7 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
show:	单击“相机”按钮, 弹出相机
stillImage	单击拍照按钮, 实现拍照
save:	照片保存
captureScreen	实现截图
saveScreenshotToPhotosAlbum:	照片保存在相册中
refreshImage	刷新照片
imagePickerController:didFinishPickingMediaWithInfo:	实现将照片显示在滚动视图中

相机的设置需要使用 viewDidLoad 和 show:方法实现。这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, show:方法实现当单击“相机”按钮, 弹出自定义相机。程序代码如下:

```

- (IBAction)show:(id)sender {
    imagePickerController = [[UIImagePickerController alloc] init];
    imagePickerController.delegate=self;                //设置委托
    imagePickerController.allowsEditing = NO;
    imagePickerController.sourceType = UIImagePickerControllerSourceType
    Camera;
    [self presentViewController:imagePickerController animated:YES comp
    letion:nil];
    UIImageView *vv=[UIImageView alloc]initWithFrame:imagePickerController.view.frame];
    UIToolbar *controlView = [[UIToolbar alloc] initWithFrame:CGRectMake(0,
    self.view.frame.size.height-88, self.view.frame.size.width, 60)];
    //自动调整位置和尺寸
    controlView.autoresizingMask = UIViewAutoresizingFlexibleTopMargin |
    UIViewAutoresizingFlexibleWidth;
    //设置拍照按钮
    UIButton *cameraBtn = [UIButton buttonWithType:UIButtonTypeCustom];
    cameraBtn.frame = CGRectMake(0, 0, 60, 60);           //设置框架

```



```

cameraBtn.showsTouchWhenHighlighted = YES;
[cameraBtn setImage:GET_IMAGE(@"1.png", nil) forState:UIControlStateNormal];
[cameraBtn addTarget:self action:@selector(stillImage) forControlEvents:UIControlEventTouchUpInside]; //添加对象
UIBarButtonItem *takePicItem = [[UIBarButtonItem alloc] initWithCustomView:cameraBtn];
UIBarButtonItem *spItem = [[UIBarButtonItem alloc] initWithBarButtonSystemItem:UIBarButtonSystemItemFlexibleSpace target:nil action:nil];
NSArray *items = [NSArray arrayWithObjects:spItem, takePicItem, spItem, nil];
[controlView setItems:items]; //设置条目
imagePickerController.showsCameraControls = NO;
[vv addSubview:controlView];
//添加水印
UIImageView *iv=[[UIImageView alloc] initWithFrame:CGRectMake(130, 350, 200, 100)];
iv.image=[UIImage imageNamed:@"2.png"]; //设置图像
[vv addSubview:iv];
imagePickerController.cameraOverlayView = vv;
}

```

照相功能需要使用 `stillImage` 方法实现。保存照片需要使用 `save:`、`captureScreen`、`saveScreenshotToPhotosAlbum:`、`refreshImage`、`imagePickerController:didFinishPickingMediaWithInfo:`方法实现。其中，`refreshImage` 方法实现对照片的刷新。程序代码如下：

```

- (void)refreshImage
{
    scrollView.hidden=NO;
    //移除所有旧的子 view
    for (UIView *subView in scrollView.subviews) {
        [subView removeFromSuperview];
    }
    scrollView.contentSize = CGSizeMake(0, 450); //设置可滚动区域的尺寸
    for (int i = 0; i < imageArray.count; i++) {
        UIImageView *iv = [[UIImageView alloc] initWithImage:[imageArray objectAtIndex:i]];
        iv.contentMode = UIViewContentModeScaleAspectFit; //设置显示模式
        iv.frame = CGRectMake(0, 0, 320, 450);
        [scrollView addSubview:iv];
        //添加水印
        UIImageView *ivm=[[UIImageView alloc] initWithFrame:CGRectMake(130, 360, 200, 100)];
        ivm.backgroundColor=[UIColor clearColor]; //设置背景颜色
        ivm.image=[UIImage imageNamed:@"2.png"];
        [scrollView addSubview:ivm];
    }
    [imageArray removeAllObjects]; //清空数组
    [scrollView setContentOffset:CGPointZero];
}

```

`saveScreenshotToPhotosAlbum:`方法实现将照片保存在相册中。程序代码如下：

```

- (void)saveScreenshotToPhotosAlbum:(UIView *)view
{
    UIImageWriteToSavedPhotosAlbum([self captureScreen], nil, nil, nil); //保存照片
}

```

captureScreen 方法实现截图功能。程序代码如下：

```
- (UIImage *) captureScreen {
    UIGraphicsBeginImageContext(self.view.bounds.size);
    [self.view.layer renderInContext:UIGraphicsGetCurrentContext()];
    UIImage *viewImage = UIGraphicsGetImageFromCurrentImageContext();
    //创建图形上下文

    UIGraphicsEndImageContext();
    CGImageRef imageRef = viewImage.CGImage;
    CGRect rect =CGRectMake(0, 66, 320, 450); //这里可以设置想要截图的区域
    CGImageRef imageRefRect =CGImageCreateWithImageInRect(imageRef, rect);
    UIImage *sendImage = [[UIImage alloc] initWithCGImage:imageRefRect];
    return sendImage;
}
```

【代码解析】

本实例关键功能是水印的添加以及单击 Save 按钮后将照片保存。下面依次讲解这两个知识点。

1. 水印的添加

在本实例中水印的添加需要使用 UIImagePickerController 的 cameraOverlayView 属性实现，代码如下：

```
imagePickerController.cameraOverlayView = vv;
```

其中，vv 是空白视图，在其中放置有图像视图用来显示水印以及拍照时使用的按钮。

2. 单击Save按钮后将照片保存

UIImageWriteToSavedPhotosAlbum 函数的功能是将指定的图像保存在相册中。其语法形式如下：

```
void UIImageWriteToSavedPhotosAlbum (
    UIImage *image,
    id completionTarget,
    SEL completionSelector,
    void *contextInfo
);
```

其中，参数说明如下：

- ❑ UIImage *image 表示指定的图像；
- ❑ id completionTarget 表示对象；
- ❑ SEL completionSelector 表示方法对应的选择器；
- ❑ void *contextInfo 表示在回调中可选择传入的数据。

在本实例中就使用了 UIImageWriteToSavedPhotosAlbum 函数将带有水印的照片保存在相册中，代码如下：

```
UIImageWriteToSavedPhotosAlbum([self captureScreen], nil, nil, nil);
```

其中，参数说明如下：

- ❑ [self captureScreen]表示指定的图像；
- ❑ 第 1 个 nil 表示没有对象；

- ❑ 第 2 个 nil 表示没有方法对应的选择器；
- ❑ 第 3 个 nil 表示在回调中没有可选择传入的数据。

实例 108 QQ 聊天视频效果

【实例描述】

在 QQ 聊天中经常可以看到，自己的 QQ 视频显示在界面的右下角。在本实例中就为读者实现 QQ 添加中的视频效果，让自己的视频显示在右下角。运行效果如图 10.8 所示。

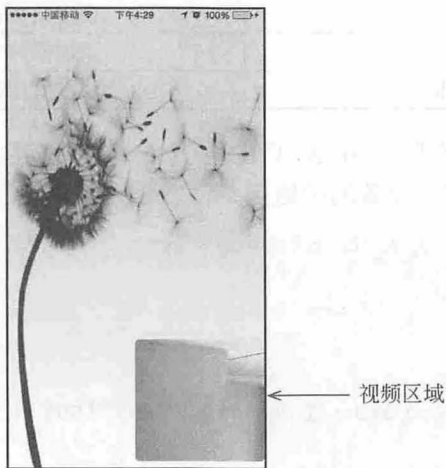


图 10.8 运行效果

【实现过程】

- (1) 创建一个项目，命名为“QQ 聊天视频效果”。
- (2) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 AVFoundation.framework 到创建的项目中。
- (4) 创建一个基于 UIView 类的 View 类。
- (5) 打开 View.h 文件，编写代码，实现头文件、定义行的数据类型、实例变量、属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
//定义新的数据类型
typedef enum{
    IPHONE3x5 = 0,
    IPHONE4 = 1,
} DeviceType;
@interface View : UIView
{
    CGPoint offset;
}
//属性
@property (nonatomic, assign) AVCaptureDevicePosition cameraType;
@property (nonatomic, strong) AVCaptureSession *session;
@property (nonatomic, assign) BOOL draggable;
```

```
//方法
-(id)initWithDeviceType:(DeviceType)type;
-(void)startFaceCam;
@end
```

(6) 打开 View.m 文件，编写代码，实现自定义视频的显示。使用的方法如表 10-8 所示。

表 10-8 View.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
initWithDeviceType:	通过设备类型进行初始化
setDefaults	设置默认设置
startFaceCam	添加摄像头
CameraIfAvailable	获取摄像头设备

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，initWithDeviceType:方法实现对设备的初始化。程序代码如下：

```
-(id)initWithDeviceType:(DeviceType)type
{
    CGRect frame;
    switch (type)
    {
        case 0:
            frame = CGRectMake(160, 410, 160, 150); //设置框架
            break;
        case 1:
            frame = CGRectMake(190, 380, 120, 160); //设置框架
            break;
    }
    self = [super initWithFrame:frame];
    if (self) {
        [self setDefaults]; //调用 setDefaults 方法
    }
    return self;
}
```

startFaceCam 方法实现视频的显示，即摄像头设置的添加功能。程序代码如下：

```
-(void)startFaceCam {
    AVCaptureDevice *device = [self CameraIfAvailable];
    //判断设置是否可用
    if (device) {
        //如果设置可用
        if (!session) {
            session = [[AVCaptureSession alloc] init]; //创建对象 session
        }
        session.sessionPreset = AVCaptureSessionPresetMedium;
        NSError *error = nil;
        AVCaptureDeviceInput *input = [AVCaptureDeviceInput deviceInput
            WithDevice:device error:&error];
        //判断是否存在输入设备
        if (!input) {
            NSLog(@"ERROR: trying to open camera: %@", error);
        } else {
```



```

//判断 session 是否可以添加输入设置
if ([session canAddInput:input]) {
    [session addInput:input]; //添加输入设置
    //创建并设置预览图层 (用来显示视频)
    AVCaptureVideoPreviewLayer *captureVideoPreviewLayer =
    [[AVCaptureVideoPreviewLayer alloc] initWithSession:session];
    captureVideoPreviewLayer.frame = self.bounds; //设置框架
    captureVideoPreviewLayer.videoGravity = AVLayerVideoGravity
    ResizeAspectFill;
    [self.layer addSublayer:captureVideoPreviewLayer];
    //添加图层对象
    [session startRunning];
} else {
    NSLog(@"Couldn't add input");
}
} else {
    //如果设置不可用
    NSLog(@"Camera not available");
}
}

```

CameraIfAvailable 方法实现摄像头设置的获取功能。程序代码如下:

```

-(AVCaptureDevice *)CameraIfAvailable
{
    NSArray *videoDevices = [AVCaptureDevice devicesWithMediaType:
    AVMediaTypeVideo];
    AVCaptureDevice *captureDevice = nil;
    //遍历
    for (AVCaptureDevice *device in videoDevices)
    {
        //判断设备的位置是否等于 cameraType
        if (device.position == self.cameraType)
        {
            captureDevice = device;
            break;
        }
    }
    //判断 captureDevice 是否为空
    if (!captureDevice) {
        captureDevice = [AVCaptureDevice defaultDeviceWithMediaType:
        AVMediaTypeVideo];
    }
    return captureDevice;
}

```

(7) 打开 ViewController.h 文件, 编写代码, 实现头文件以及对象的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "View.h"
@interface ViewController : UIViewController
{
    View *viewer;
}
@end

```

(8) 打开 Main.storyboard 文件, 从视图中拖动 Image View 图像视图到设计界面中,

将 Image 属性设置为 1.jpg。

(9) 打开 ViewController.m 文件，编写代码，实现 View 类对象的创建以及添加。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    viewer = [[View alloc] initWithDeviceType:IPHONE3x5];
    [viewer startFaceCam];
    [self.view addSubview:viewer]; //添加对象
}
```

【代码解析】

本实例关键功能是改变视频的区域大小。下面就是这个知识点的详细讲解。

AVCaptureVideoPreviewLayer 类是一个预览图层，用来显示视频。在本实例中就使用了此类实现了视频的显示。它的区域大小可以使用 frame 属性进行设置。程序代码如下：

```
AVCaptureVideoPreviewLayer *captureVideoPreviewLayer = [[AVCaptureVideoPreviewLayer alloc] initWithSession:session];
captureVideoPreviewLayer.frame = self.bounds;
```

实例 109 iOS 7 手电筒实现

【实例描述】

在 iOS 7 中，添加了一个手电筒的功能。这个手电筒的功能其实就是使用了闪光灯功能实现的。在本实例中就为各位读者来实现这个手电筒的功能。当用户打开开关时，闪光灯随即打开；当开关关闭时，闪光灯关闭。运行效果如图 10.9 所示。

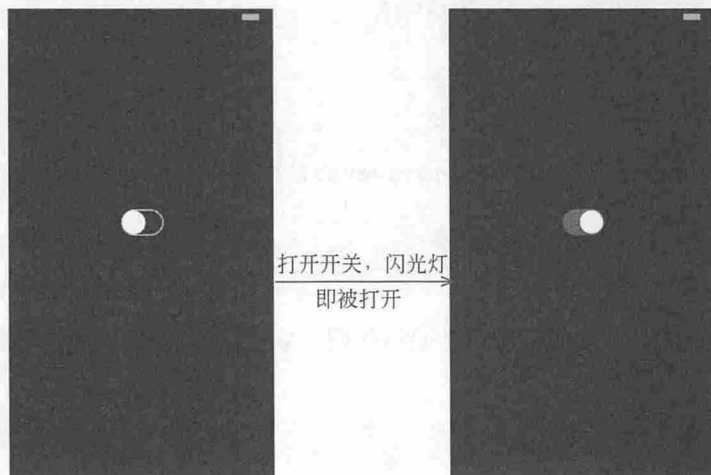


图 10.9 运行效果

【实现过程】

(1) 创建一个项目，命名为“iOS 7 手电筒实现”。

(2) 添加 AVFoundation.framework 到创建的项目中。

(3) 打开 ViewController.h 文件, 编写代码, 实现头文件、对象、实例变量以及属性的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface ViewController : UIViewController{
    UISwitch                *flashlightButton;           //声明关于开关的对象
    AVCaptureSession        *AVSession;
    BOOL                    flashlightOn;
}
@property (nonatomic, retain) AVCaptureSession *AVSession;
@end
```

(4) 打开 Main.storyboard 文件, 将设计界面的背景设置为黑色。

(5) 打开 ViewController.m 文件, 编写代码, 实现开关手电筒的功能。使用的方法如表 10-9 所示。

表 10-9 ViewController.m文件中方法总结

方 法	功 能
toggleFlashlight	切换闪光灯
loadView	加载视图
buttonPressed:	开关的切换

toggleFlashlight 方法实现切换闪光灯的功能, 即打开闪光灯、关闭闪光灯。程序代码如下:

```
- (void)toggleFlashlight
{
    AVCaptureDevice *device = [AVCaptureDevice defaultDeviceWithMediaType:
    AVMediaTypeVideo];
    //判断设备的闪光灯是否是关着的
    if (device.torchMode == AVCaptureTorchModeOff){
        AVCaptureSession *session = [[AVCaptureSession alloc] init];
        //创建并设置输入输出设备
        AVCaptureDeviceInput *input = [AVCaptureDeviceInput deviceInputWith
        Device:device error:nil];
        [session addInput:input]; //添加属性设置
        AVCaptureVideoDataOutput *output = [[AVCaptureVideoDataOutput alloc]
        init];
        [session addOutput:output]; //添加输出设置
        [session beginConfiguration];
        [device lockForConfiguration:nil];
        device.torchMode=AVCaptureTorchModeOn; //开闪光灯
        [device unlockForConfiguration];
        [session commitConfiguration];
        [session startRunning]; //开始运行
        [self setAVSession:session];
    }else{
        [AVSession stopRunning]; //开始运行
        AVSession = nil;
    }
}
```

loadView 方法实现视图的加载, 即创建开关控件。程序代码如下:

```
- (void)loadView
{
    [self setView:[[UIView alloc] initWithFrame:[UIScreen mainScreen]
    applicationFrame]] ;
    AVCaptureDevice *device = [AVCaptureDevice defaultDeviceWithMediaType:
    AVMediaTypeVideo];
    //判断是否有闪光灯
    if ([device hasTorch] == YES)
    {
        //创建并设置开关对象
        flashlightButton = [[UISwitch alloc] initWithFrame:CGRectMake(136,
        242, 49, 31)];
        flashlightButton.onTintColor=[UIColor redColor];           //着色
        flashlightButton.on=NO;
        [flashlightButton addTarget:self action:@selector(buttonPressed:)
        forControlEvents: UIControlEventTouchUpInside];           //添加动作
        [[self view] addSubview:flashlightButton];
    }
}
```

buttonPressed:方法实现开关的切换。程序代码如下：

```
- (void)buttonPressed:(UIButton *)button
{
    //判断 flashlightOn 是否为 NO
    if (flashlightOn == NO){
        flashlightOn = NO;
    }else {
        flashlightOn = YES;
    }
    [self toggleFlashlight];           切换闪光灯
}
```

【代码解析】

本实例关键功能是闪光灯的打开和关闭。下面就是这个知识点的详细讲解。

AVCaptureDevice 的 torchMode 属性的功能就是对闪光灯的打开和关闭进行设置，其语法形式如下：

```
@property(nonaatomic) AVCaptureTorchMode torchMode;
```

其中，它的参数设置如表 10-10 所示。

表 10-10 参数

参 数	功 能
AVCaptureTorchModeOff	关闭闪光灯
AVCaptureTorchModeOn	打开闪光灯
AVCaptureTorchModeAuto	自动化

实例 110 三 连 拍

【实例描述】

在 iOS 自带的相机应用中是不存在连拍功能的。在本实例中就为读者弥补它的功能，

为它实现三连拍的效果。当用户单击“拍照”按钮，就会弹出自定义的相机。当用户将开关控件移到开时，单击“拍照”按钮，会就实现三连拍的功能。当用户单击“完成”按钮，拍摄的照片就会显示在滚动视图中。运行效果如图 10.10 所示。

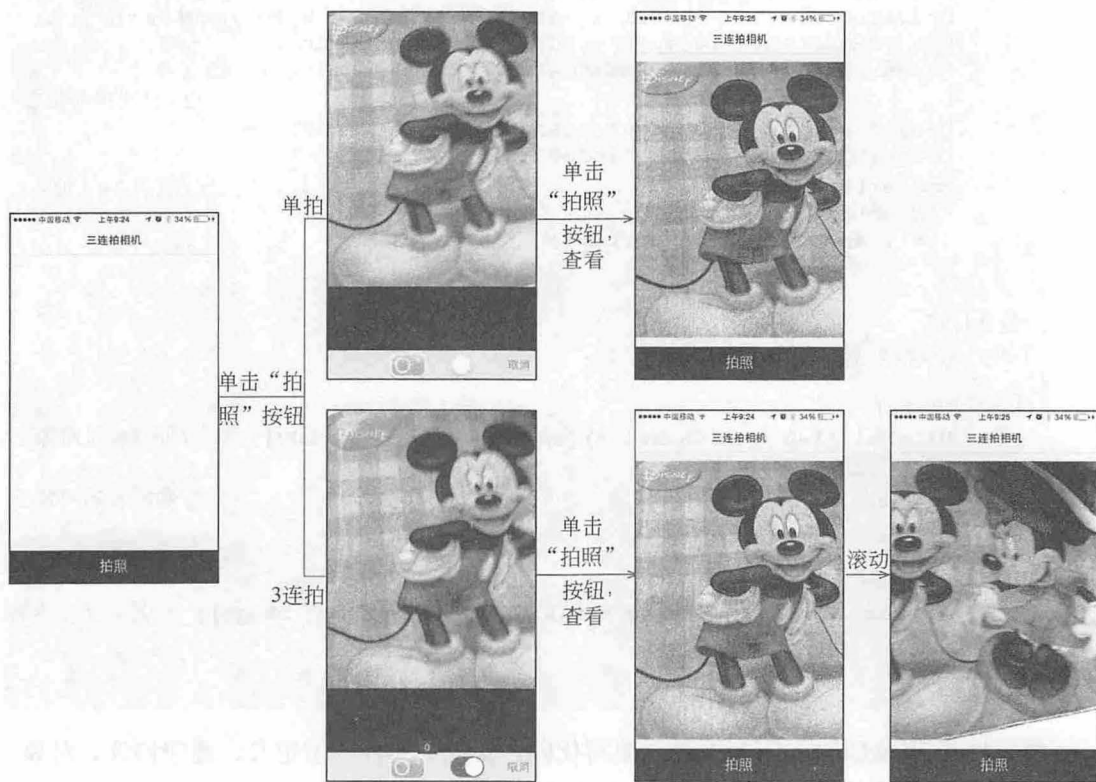


图 10.10 运行效果

【实现过程】

- (1) 创建一个项目，命名为“三连拍”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 AVFoundation.framework 到创建的项目中。
- (4) 创建一个基于 UIButton 类的分类 Badge。
- (5) 打开 UIButton+Badge.h 文件，编写代码，实现方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface UIButton (Badge)
//方法
- (void)badgeNumber: (int)number;
- (void)setBadge: (int)badge;
@end
```

- (6) 打开 UIButton+Badge.m 文件，编写代码，实现方法的定义。程序代码如下：

```
//创建并设置标签对象
- (void)badgeNumber: (int)number
{
    if (self) {
        //创建并设置标签对象
```

```

UILabel *badgerLab = [[UILabel alloc] initWithFrame:CGRectMake(0, 0,
30, 20)];
badgerLab.layer.cornerRadius = 10.0;
badgerLab.backgroundColor = [UIColor darkGrayColor];
badgerLab.textColor = [UIColor whiteColor]; //设置文本颜色
badgerLab.text = [NSString stringWithFormat:@"%d", number];
badgerLab.textAlignment = NSTextAlignmentCenter;
badgerLab.center = CGPointMake(self.frame.size.width, 0); //设置中心位置

badgerLab.font = [UIFont systemFontOfSize:14];
badgerLab.adjustsFontSizeToFitWidth = YES;
badgerLab.tag = 100; //设置 tag 值
badgerLab.hidden = YES;
[self addSubview:badgerLab];
}
}
//设计标记
- (void)setBadge:(int)badge
{
    if (self) {
        UILabel *lab = (UILabel *)[self viewWithTag:100]; //创建标签对象
        if (badge == -1) {
            lab.hidden = YES; //隐藏标签对象
        }
        else {
            lab.hidden = NO;
            lab.text = [NSString stringWithFormat:@"%d", badge]; //显示文本内容
        }
    }
}
}

```

(7) 打开 ViewController.h 文件，编写代码，实现头文件、宏定义、遵守协议、对象、插座变量、实例变量以及属性、动作、方法的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "UIButton+Badge.h"
#define GET_IMAGE( __NAME__, __TYPE__) [UIImage imageNamed:[NSBundle mainBundle] pathForResource:__NAME__ ofType:__TYPE__]]
@interface ViewController : UIViewController<UINavigationControllerDelegate, UIImagePickerControllerDelegate>{
    //对象
    NSMutableArray *imageArray;
    NSTimer *timer;
    IBOutlet UIScrollView *scrollView;
    //实例变量
    BOOL singleMode;
    int j;
}
@property (nonatomic, retain) UIImagePickerController *imagePickerController; //属性

//方法
- (IBAction)takePhoto:(id)sender;
- (void)setupImagePicker:(UIImagePickerControllerSourceType) sourceType;
@end

```

(8) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 10.11 所示。

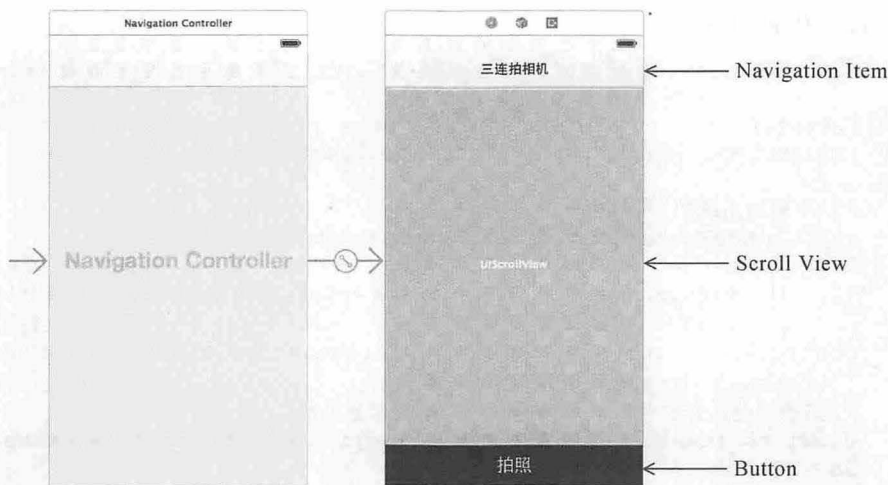


图 10.11 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 10-11 所示。

表 10-11 视图、控件设置

视图、控件	属 性 设 置	其 他
Navigation Item	Title: 三连拍相机	
Scroll View		与插座变量 scrollView 关联
Button	Title: 拍照 Font: System 22.0 Text Color: 白色 Background: 黑色	与动作 takePhoto: 关联

(9) 打开 ViewController.m 文件，编写代码，实现对相机界面的设计、照相（三连拍或者单拍）以及照片保存的功能。使用的方法如表 10-12 所示。

表 10-12 ViewController.m 文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
captureModeChanged:	改变相机拍照的模式
refreshImage	刷新照片
takePhoto:	单击“拍照”按钮，弹出相机
setupImagePicker:	设计相机界面的元素
stillImage:	单击拍照按钮后的判断
take	拍照
doneAction	单击“完成”或者“取消”按钮
imagePickerControllerDidCancel:	关闭相机应用
imagePickerController:didFinishPickingMediaWithInfo:	用户选择照片后调用，实现将照片显示在滚动视图中

相机界面的设计需要使用 viewDidLoad、setupImagePicker: 方法。这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，setupImagePicker: 方法实现对相机界面元

素的设计。程序代码如下：

```

- (void)setupImagePicker:(UIImagePickerControllerSourceType)sourceType
{
    self.imagePickerController.sourceType = sourceType;
    if (sourceType == UIImagePickerControllerSourceTypeCamera)
    {
        // 不使用系统的控制界面
        self.imagePickerController.showsCameraControls = NO;
        UIToolbar *controlView = [[UIToolbar alloc] initWithFrame: CGRectMake(0, self.view.frame.size.height-44, self.view.frame.size.width, 44)];
        // 创建工具栏对象
        controlView.autoresizingMask = UIViewAutoresizingFlexibleTopMargin
        | UIViewAutoresizingFlexibleWidth; // 自动调整位置与尺寸
        // 创建并设置开关对象，用来控制三连拍或者是单拍
        UISwitch *aswitch=[[UISwitch alloc] initWithFrame:CGRectMake(0, 0,
        25, 10)];
        aswitch.onTintColor=[UIColor redColor]; // 着色
        [aswitch addTarget:self action:@selector(captureModeChanged:)
        forControlEvents:UIControlEventTouchUpInside]; // 添加动作
        UIBarButtonItem *switchItem = [[UIBarButtonItem alloc] initWith
        CustomView:aswitch];
        // 拍照
        UIButton *cameraBtn = [UIButton buttonWithType:UIButtonTypeCustom];
        cameraBtn.frame = CGRectMake(0, 0, 60, 60); // 设置框架
        cameraBtn.showsTouchWhenHighlighted = YES;
        [cameraBtn setImage:GET_IMAGE(@"1.png", nil) forState:UIControlStateNormal];
        [cameraBtn addTarget:self action:@selector(stillImage:) forControlEvents:
        UIControlEventTouchUpInside]; // 添加动作
        [cameraBtn badgeNumber:-1];
        UIBarButtonItem *takePicItem = [[UIBarButtonItem alloc] initWith
        CustomView:cameraBtn];
        // 取消、完成
        UIBarButtonItem *doneItem = [[UIBarButtonItem alloc] initWith
        Title:@"取消" style:UIBarButtonItemStyleBordered target:self
        action: @selector(doneAction)];
        // 空 item
        UIBarButtonItem *spItem = [[UIBarButtonItem alloc] initWithBarButton
        SystemItem:UIBarButtonItemSystemItemFlexibleSpace target:nil action:nil];
        NSArray *items = [NSArray arrayWithObjects:spItem,spItem,spItem,
        takePicItem,spItem,switchItem,spItem,doneItem, nil];
        [controlView setItems:items]; // 设置条目
        self.imagePickerController.cameraOverlayView = controlView;
        controlView = nil;
    }
}

```

照相（三连拍或者单拍）功能的实现需要使用 `captureModeChanged:`、`takePhoto:`、`stillImage:`、`take:`、`doneAction` 方法。其中，`captureModeChanged:` 方法实现对相机模式改变的控制（即实现三连拍和单拍的切换设置）。程序代码如下：

```

- (IBAction)captureModeChanged:(id)sender
{
    UISwitch *modeSwitch = (UISwitch *)sender;
    singleMode = ![modeSwitch.isOn];
    UIToolbar *view = (UIToolbar *)self.imagePickerController.camera
    OverlayView; // 创建工具栏对象
}

```

```

UIBarButtonItem *cameraItem = [[view items] objectAtIndex:3];
[(UIButton *)cameraItem.customView setBadge:singleMode? -1:0];
}

```

stillImage:方法实现单击拍照按钮后的判断,判断当前相机的模式是否为单拍模式。程序代码如下:

```

- (void)stillImage:(id) sender
{
    //判断 singleMode 是否为 YES
    if (singleMode==YES ) {
        [self.imagePickerController takePicture];
    }else{
        j=1;
        timer=[NSTimer scheduledTimerWithTimeInterval:1.0 target:self
selector:@selector(take) userInfo:nil repeats:YES];    //创建时间定时器
    }
}

```

take 方法实现拍照功能。

```

- (void)take{
    j++;
    if (j>4) {
        [timer invalidate];    //让定时器失效
    }else{
        [self.imagePickerController takePicture];    //照相
    }
}

```

照片保存需要使用 refreshImage、imagePickerControllerDidCancel、imagePickerController:didFinishPickingMediaWithInfo:方法。其中,imagePickerController:didFinishPickingMediaWithInfo:方法实现将照片显示在滚动视图中。程序代码如下:

```

- (void)imagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    //保存相片到数组,这种方法不可取,会占用过多内存
    [imageArray addObject:[info objectForKey:UIImagePickerController
OriginalImage]];
    //判断是否为单张
    if (singleMode) {
        [picker dismissViewControllerAnimated:YES completion:nil];
        [self refreshImage];    //刷新图像
    }
    else if (imageArray.count == 1) {
        //多张拍摄,不必每次都执行
        UIBarButtonItem *flashItem = [[(UIToolbar *)self.imagePickerController.cameraOverlayView items] lastObject];
        flashItem.title = @"完成";    //设置标题
    }
    UIToolbar *view = (UIToolbar *)self.imagePickerController.cameraOverlayView;
    UIBarButtonItem *cameraItem = [[view items] objectAtIndex:3];
    [(UIButton *)cameraItem.customView setBadge:imageArray.count];
}

```

【代码解析】

在本实例中三连拍的实现,首先需要判断当前拍照的模式是单拍模式还是三连拍模

式。程序代码如下：

```
if (singleMode==YES ) {  
    [self.imagePickerController takePicture];  
}else{  
    j=1;  
    timer=[NSTimer scheduledTimerWithTimeInterval:1.0 target:self  
        selector:@selector(take) userInfo:nil repeats:YES];  
}
```

如果是连拍模式就会调用定时器中的 take 方法，此方法实现了在每隔 1.0 秒后的拍照功能。程序代码如下：

```
j++;  
if (j>4) {  
    [timer invalidate];  
}else{  
    [self.imagePickerController takePicture];  
}
```

实例 111 条形码/二维码的扫描

【实例描述】

iOS 7 问世以后，带来了很多的新特性，其中一项就是对条形码以及二维码等的扫描功能，摆脱使用第三方类的命运。本实例就为读者实现 iOS 7 的这一新功能——条形码/二维码的扫描。运行效果如图 10.12 所示。

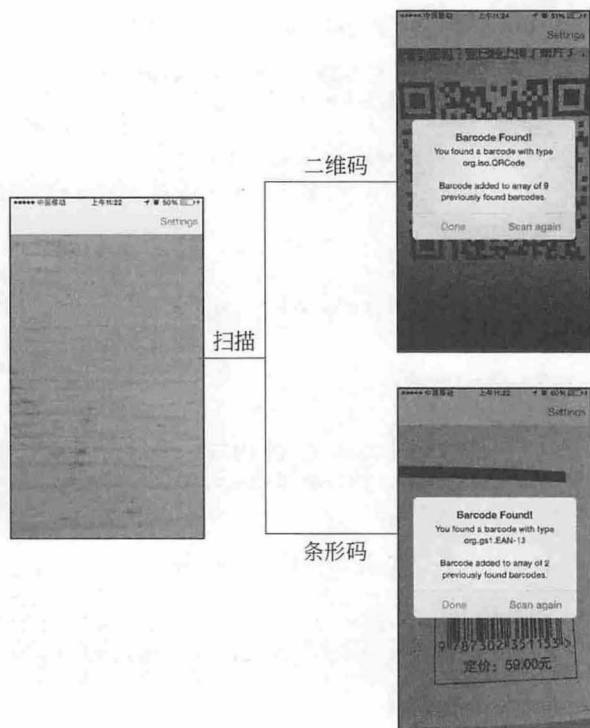


图 10.12 运行效果

【实现过程】

- (1) 创建一个项目，命名为“条形码/二维码的扫描”。
- (2) 添加 AVFoundation.framework 到创建的项目中。
- (3) 创建一个基于 NSObject 类的 Barcode 类。
- (4) 打开 Barcode.h 文件，编写代码，实现头文件、属性以及方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
#import AVFoundation;
@interface Barcode : NSObject
//属性
@property (nonatomic, strong) AVMetadataMachineReadableCodeObject
*metadataObject;
@property (nonatomic, strong) NSString * barcodeType;
@property (nonatomic, strong) NSString * barcodeData;
@property (nonatomic, strong) UIBezierPath *cornersPath;
@property (nonatomic, strong) UIBezierPath *boundingBoxPath;
//方法
+ (Barcode *)processMetadataObject:(AVMetadataMachineReadableCodeObject*)
code;
- (NSString *) getBarcodeType; //获取条形码类型
- (NSString *) getBarcodeData;
- (void) printBarcodeData;
@end
```

(5) 打开 Barcode.m 文件，编写代码，实现对条形码的获取。使用的方法如表 10-13 所示。

表 10-13 Barcode.m文件中方法总结

方 法	功 能
processMetadataObject:	获取条形码
getBarcodeType	获取条形码类型
getBarcodeData	获取条形码数据
printBarcodeData	输出条形码数据

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。processMetadataObject: 方法实现对条形码的获取。程序代码如下：

```
+ (Barcode *)processMetadataObject: (AVMetadataMachineReadableCode
Object*)code
{
    //创建并设置条形码对象
    Barcode * barcode=[[Barcode alloc]init];
    barcode.barcodeType = [NSString stringWithString:code.type];
    barcode.barcodeData = [NSString stringWithString:code.stringValue];
    //设置数据
    barcode.metadataObject = code;
    CGMutablePathRef cornersPath = CGPathCreateMutable();
    CGPoint point;
    //将字典转换为 CGSize
    CGPointMakeWithDictionaryRepresentation((CFDictionaryRef)code.corners[0],
    &point);
    CGPathMoveToPoint(cornersPath, nil, point.x, point.y); //设置开始点
```



```

//画线
for (int i = 1; i < code.corners.count; i++) {
    CGPointMakeWithDictionaryRepresentation( (CFDictionaryRef)code.
        corners[i], &point);
    CGPathAddLineToPoint(cornersPath, nil, point.x, point.y); //设置结束点
}
CGPathCloseSubpath(cornersPath); //关闭子路径
barcode.cornersPath = [UIBezierPath bezierPathWithCGPath:cornersPath];
CGPathRelease(cornersPath);
barcode.boundingBoxPath = [UIBezierPath bezierPathWithRect:code.bounds];
return barcode;
}

```

(6) 创建一个基于 UIViewController 类的 SettingsViewController 类。

(7) 打开 SettingsViewController.h 文件，编写代码，实现协议以及属性的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
//协议
@protocol SettingsDelegate <NSObject>
@required
- (void) settingsChanged:(NSMutableArray *)allowedTypes;
@end
@interface SettingsViewController : UIViewController
//属性
@property (weak) id <SettingsDelegate> delegate;
.....
@property (strong, nonatomic) IBOutletCollection(UISwitch) NSArray
*barcodeSwitches;
@end

```

(8) 打开 SettingsViewController.m 文件，编写代码，实现对条形码类型的设置。程序代码如下：

```

//视图加载后调用，实现初始化
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view.
    //设置按钮
    self.saveButton.layer.cornerRadius = 8.0f;
    self.saveButton.layer.borderColor = [UIColor colorWithRed:0.0
        green:122.0/255.0 blue:1.0 alpha:1.0].CGColor; //设置边框颜色
    self.saveButton.layer.borderWidth=1.5f;
    self.barcodeTypes = [NSMutableArray new];
    //为数组添加对象
    [self.barcodeTypes addObject:@"org.iso.QRCode"];
    [self.barcodeTypes addObject:@"org.iso.PDF417"];
    [self.barcodeTypes addObject:@"org.gs1.UPC-E"];
    [self.barcodeTypes addObject:@"org.iso.Aztec"];
    [self.barcodeTypes addObject:@"org.iso.Code39"];
    [self.barcodeTypes addObject:@"org.iso.Code39Mod43"];
    [self.barcodeTypes addObject:@"org.gs1.EAN-13"];
    [self.barcodeTypes addObject:@"org.gs1.EAN-8"];
    [self.barcodeTypes addObject:@"com.intermec.Code93"];
    [self.barcodeTypes addObject:@"org.iso.Code128"];
}
//单击 Save 按钮后，改变 barcodeTypes 中的内容

```

```

- (IBAction)saveButtonPressed:(id)sender {
    self.allowedBarcodeTypes = [NSMutableArray new];
    for (UISwitch * sw in self.barcodeSwitches){
        //判断开关是否为开
        if (sw.isOn){
            [self.allowedBarcodeTypes addObject:[self.barcodeTypes objectAtIndex:sw.tag]]; //添加对象
        }
    }
    [self.delegate settingsChanged:self.allowedBarcodeTypes];
    [self.navigationController popViewControllerAnimated:YES];
}

```

(9) 创建一个基于 UIViewController 类的 ScannerViewController 类。

(10) 打开 ScannerViewController.h 文件，编写代码，实现头文件、遵守协议、对象、实例变量以及属性的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
//头文件
#import "SettingsViewController.h"
#import <AVFoundation/AVFoundation.h>
@interface ScannerViewController : UIViewController<UIAlertViewDelegate, SettingsDelegate>{
    //对象
    AVCaptureSession *_captureSession;
    AVCaptureDevice *_videoDevice;
    AVCaptureDeviceInput *_videoInput;
    AVCaptureVideoPreviewLayer *_previewLayer;
    AVCaptureMetadataOutput *_metadataOutput;
    BOOL _running; //布尔类型的实例变量
}
//属性
@property (strong, nonatomic) NSMutableArray * allowedBarcodeTypes;
@property (strong, nonatomic) NSMutableArray * foundBarcodes;
@property (weak, nonatomic) IBOutlet UIView *previewView;
@property (strong, nonatomic) SettingsViewController * settingsVC;
@end

```

(11) 打开 ScannerViewController.m 文件，编写代码，实现扫描的功能。使用的方法如表 10-14 所示。

表 10-14 ScannerViewController.m 文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
viewDidAppear:	视图已完全过渡到屏幕上时调用
viewWillDisappear:	视图即将消失时调用
setupCaptureSession	设置 CaptureSession 捕获会话
startRunning	启动捕获会话对象以启动数据流
stopRunning	结束捕获会话对象以结束数据流
applicationWillEnterForeground:	程序从后台将要重新回到前台时候调用，实现调用 startRunning 方法
applicationDidEnterBackground:	程序被推送到后台的时候调用，实现调用 stopRunning 方法
captureOutput:didOutputMetadataObjects:	获取视频帧
validBarcodeFound:	发现条形码时调用

续表

方 法	功 能
showBarcodeAlert:	显示警告视图
alertView:clickedButtonAtIndex:	警告视图的响应
settingsChanged:	改变 barcodeTypes 中的内容

其中，`setupCaptureSession` 方法实现捕获会话的创建。程序代码如下：

```
- (void)setupCaptureSession {
    //判断 CaptureSession 对象是否存在
    if (!_captureSession)
        return;
    _videoDevice = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];
    //实例化对象

    //判断 _videoDevice 是否为空
    if (!_videoDevice) {
        NSLog(@"No video camera on this device!");
        return;
    }
    //创建 CaptureSession 对象
    _captureSession = [[AVCaptureSession alloc] init];
    _videoInput = [[AVCaptureDeviceInput alloc] initWithDevice:_videoDevice
        error:nil];
    //判断 session 是否可以添加输入设置
    if ([_captureSession canAddInput:_videoInput]) {
        [_captureSession addInput:_videoInput];
        //添加输入设置
    }
    //创建并设置预览图层（用来显示视频）
    _previewLayer = [[AVCaptureVideoPreviewLayer alloc] initWithSession:
        _captureSession];
    _previewLayer.videoGravity = AVLayerVideoGravityResizeAspectFill;
    _metadataOutput = [[AVCaptureMetadataOutput alloc] init];
    //实例化对象 _metadataOutput
    dispatch_queue_t metadataQueue =
        dispatch_queue_create("com.1337labz.featurebuild.metadata", 0);
    [_metadataOutput setMetadataObjectsDelegate:self queue:metadataQueue];
    //判断
    if ([_captureSession canAddOutput:_metadataOutput]) {
        [_captureSession addOutput:_metadataOutput];
        //添加输出设置
    }
}
```

`captureOutput:didOutputMetadataObjects:fromConnection:` 方法实现获取视频帧的功能。

程序代码如下：

```
- (void)captureOutput:(AVCaptureOutput *)captureOutput didOutputMetadataObjects:
    (NSArray *)metadataObjects fromConnection:(AVCaptureConnection *)connection
{
    //枚举
    [metadataObjects enumerateObjectsUsingBlock:^(AVMetadataObject *obj,
        NSUInteger idx, BOOL *stop)
```

```

{
    //判断 obj 是否在 AVMetadataMachineReadableCodeObject 中
    if ([obj isKindOfClass: [AVMetadataMachineReadableCodeObject class]])
    {
        AVMetadataMachineReadableCodeObject *code = (AVMetadataMachine
        ReadableCodeObject*)
        [_previewLayer transformedMetadataObjectForMetadataObject:obj];
        Barcode * barcode = [Barcode processMetadataObject:code];
                                                                    //创建条形码对象

        //遍历
        for(NSString * str in self.allowedBarcodeTypes){
            if([barcode.getBarcodeType isEqualToString:str]){
                [self validBarcodeFound:barcode];
                return;
            }
        }
    }
}
};
}

```

showBarcodeAlert:方法实现显示警告视图的功能。程序代码如下:

```

- (void) showBarcodeAlert:(Barcode *)barcode{
    dispatch_async( dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_
    DEFAULT, 0), ^{
        //创建并设置字符串对象
        NSString * alertMessage = @"You found a barcode with type ";
        alertMessage = [alertMessage stringByAppendingString:[barcode
        getBarcodeType]];
        alertMessage = [alertMessage stringByAppendingString:@"\n\nBarcode
        added to array of "];
        alertMessage = [alertMessage stringByAppendingString:[NSString string
        WithFormat:@"%lu", (unsigned long) [self.foundBarcodes count]-1]];
        alertMessage = [alertMessage stringByAppendingString:@" previously
        found barcodes."];
        //创建警告视图对象
        UIAlertView *message = [[UIAlertView alloc] initWithTitle:@"Barcode
        Found!" message:alertMessage delegate:self cancelButtonTitle:@"Done"
        otherButtonTitles:@"Scan again",nil];
        dispatch_async(dispatch_get_main_queue(), ^{
            [message show];
                                                                    //显示警告视图
        });
    });
}

```

(12) 打开 Main.storyboard 文件, 拖动 Navigation Controller 导航控制器到画布中, 将此控制器关联的根视图设置为 View Controller 视图控制器。将此视图控制器的 Class 设置为 ScannerViewControlle。这时视图控制器变为了 Scanner View Controlle。从视图库中拖动 View Controller 视图控制器到画布中。将 Class 设置为 SettingsViewControllor。这时新增的 View Controller 视图控制器就变为了 Settings View Controller 视图控制器。对画布中的视图控制器进行设计, 效果如图 10.13 所示。

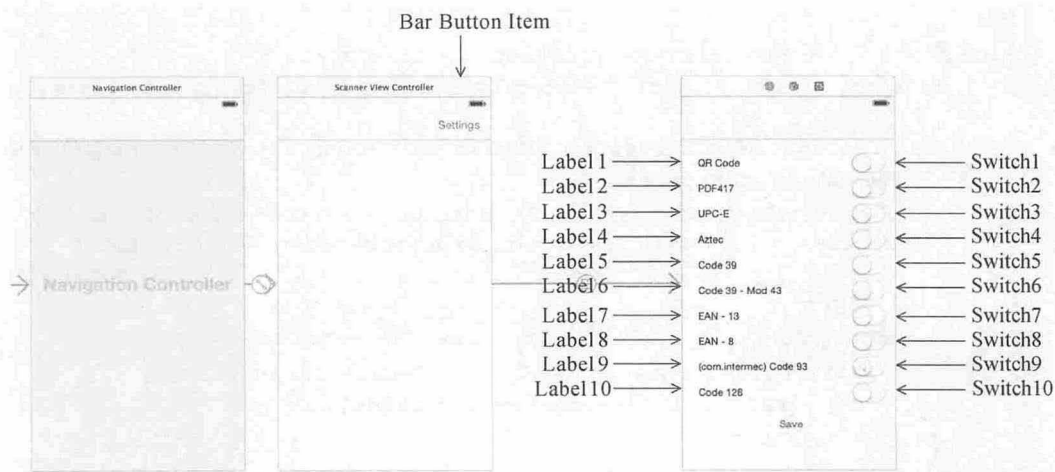


图 10.13 画布效果

需要添加的视图、控件以及对它们的设置如表 10-15 所示。

表 10-15 视图、控件设置

视图、控件	属 性 设 置	其 他
Scanner View Controllere 视图控制器		
Bar Button Item	Title: Settings	与 Settings View Controller 视图控制器
设计界面		与插座变量 previewView 关联
Settings View Controller 视图控制器		
Label1	Text: QR Code Font: System 15.0	
Label2	Text: PDF417 Font: System 15.0	
Label3	Text: UPC-E Font: System 15.0	
Label4	Text: Aztec Font: System 15.0	
Label5	Text: Code 39 Font: System 15.0	
Label6	Text: Code 39 - Mod 43 Font: System 15.0	
Label7	Text: EAN - 13 Font: System 15.0	
Label8	Text: EAN - 8 Font: System 15.0	
Label9	Text: (com.intermec) Code 93 Font: System 15.0	
Label10	Text: Code 128 Font: System 15.0	
Switch1	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联

续表

视图、控件	属 性 设 置	其 他
Switch2	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch3	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch4	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch5	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch6	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch7	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch8	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch9	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Switch10	State: Off On Tint: 蓝色	与插座变量 barcodeSwitches 关联
Button	Title: Save Font: System Bold 16.0	与插座变量 saveButton 关联 与动作 saveButtonPressed:关联

【代码解析】

本实例关键功能是条形码以及二维码的扫描和一些条码的类型的介绍。下面依次讲解这两个知识点。

1. 条形码以及二维码的扫描

AVCaptureMetaDataOutput 类支持二维码及其他类型码识别, 在本实例中就使用了 AVCaptureMetaDataOutput 类对条形码以及二维码进行了扫描。其中创建代码如下:

```
_metadataOutput = [[AVCaptureMetadataOutput alloc] init];
dispatch_queue_t metadataQueue = dispatch_queue_create("com.1337labz.
featurebuild.metadata", 0);
[_metadataOutput setMetadataObjectsDelegate:self queue:metadataQ
```

实现扫描的代码如下:

```
- (void)captureOutput:(AVCaptureOutput *)captureOutput didOutputMetadata
Objects:(NSArray *)metadataObjects fromConnection:(AVCaptureConnection
*)connection
{
    .....
    for(NSString * str in self.allowedBarcodeTypes){
        if([barcode.getBarcodeType isEqualToString:str]){
            [self validBarcodeFound:barcode];
            return;
        }
    }
    .....
}
```

2. 条码的类型的介绍

在本实例中有很多的条形码以及二维码的类型，它们的说明如表 10-16 所示。

表 10-16 条码的类型

条码的类型	说 明
QR Code	一种矩阵码，或二维空间的条码
PDF417	一种高密度、高信息含量的便携式条码
UPC-E	美国统一代码委员会制定的一种商品条码
Aztec	适合在有限的空间上标识，一般应用在手持终端（如手机）上
Code 39	是条形码的一种，由 5 个条和分开它们的 4 条缝隙，共 9 个元素构成
EAN - 13	EAN 条码是国际物品编码协会制定的一种条码，共计 13 位代码的 EAN-13 是比较通用的，主要应用于超级市场和其他零售业
EAN - 8	EAN-8 商品条码是表示 EAN/UCC-8 商品标识代码的条码符号，由左侧空白区、起始符、左侧数据符、中间分隔符、右侧数据符、校验符、终止符、右侧空白区及供人识别字符组成
Code 93	与 39 码具有相同的字符集，但它的密度要比 39 码高，所以在面积不足的情况下，可以用 93 码代替 39 码
Code 128	高密度数据，字符串可变长，符号内含校验码的条码

实例 112 魔 术

【实例描述】

在本实例实现的功能是将手指移到后置摄像头上时，就会从帽子中出现一只可爱的小兔子；当手指离开后置摄像头，小兔子就会消失，类似于变魔术的功能。运行效果如图 10.14 所示。



图 10.14 运行效果

【实现过程】

- (1) 创建一个项目，命名为“魔术”。
- (2) 添加图像 1.jpg、2.jpg 到创建项目的 Supporting Files 文件中。
- (3) 添加 AVFoundation.framework 到创建的项目中。
- (4) 创建一个基于 NSObject 类的 Magic 类。
- (5) 打开 Magic.h 文件，编写代码，实现头文件、宏定义、对象、实例变量以及方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
#import <AVFoundation/AVFoundation.h>
//宏定义
#define NUMBER_OF_FRAME_PER_S 5
#define BRIGHTNESS_THRESHOLD 70
#define MIN_BRIGHTNESS_THRESHOLD 10
@interface Magic : NSObject
{
    AVCaptureSession *_captureSession;
    //实例变量
    int _lastTotalBrightnessValue;
    int _brightnessThreshold;
    BOOL _started;
}
//方法
- (BOOL) startCapture; //开始捕获
- (BOOL) stopCapture;
- (void) updateBrightnessThreshold: (int) pValue;
@end
```

- (6) 打开 Magic.m 文件，编写代码，实现魔术事件，即遮挡摄像头的事件。使用的方法如表 10-17 所示。

表 10-17 Magic.m 文件中方法总结

方 法	功 能
init	初始化
initMagicEvents	初始化事件
initCapture	初始化捕获
updateBrightnessThreshold:	更新亮度的值
startCapture	开始捕获
stopCapture	结束捕获
captureOutput:didOutputSampleBuffer:fromConnection:	获取视频帧
calculateLevelOfBrightness:	计算亮度的级别
searchForBackCameraIfAvailable	获取相机设备

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，captureOutput:didOutputSampleBuffer:fromConnection:方法实现获取视频帧的功能。程序代码如下：

```
- (void) captureOutput: (AVCaptureOutput *) captureOutput
didOutputSampleBuffer: (CMSampleBufferRef) sampleBuffer
fromConnection: (AVCaptureConnection *) connection
{
```

```

//为媒体数据设置一个 CMSampleBuffer 的 Core Video 图像缓存对象
CVImageBufferRef imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
//判断 pixel buffer 锁定的基地址
if (CVPixelBufferLockBaseAddress(imageBuffer, 0) == kCVReturnSuccess)
{
    //得到 pixel buffer 的基地址
    UInt8 *base = (UInt8 *)CVPixelBufferGetBaseAddress(imageBuffer);
    //得到 pixel buffer 的行字节数
    size_t bytesPerRow = CVPixelBufferGetBytesPerRow(imageBuffer);
    //得到 pixel buffer 的宽和高
    size_t width = CVPixelBufferGetWidth(imageBuffer);
    size_t height = CVPixelBufferGetHeight(imageBuffer);
    UInt32 totalBrightness = 0;
    //遍历
    for (UInt8 *rowStart = base; height; rowStart += bytesPerRow, height --)
    {
        size_t columnCount = width;
        //遍历
        for (UInt8 *p = rowStart; columnCount; p += 4, columnCount --)
        {
            UInt32 value = (p[0] + p[1] + p[2]);
            totalBrightness += value;
        }
    }
    //解锁 pixel buffer
    CVPixelBufferUnlockBaseAddress(imageBuffer, 0);
    if (_lastTotalBrightnessValue == 0)
        _lastTotalBrightnessValue = totalBrightness;
    //判断计算的亮度级别
    if ([self calculateLevelOfBrightness:totalBrightness] < _brightnessThreshold)
    {
        if ([self calculateLevelOfBrightness:totalBrightness] > MIN_BRIGHTNESS_THRESHOLD)
        {
            [[NSNotificationCenter defaultCenter] postNotificationName:@"onMagicEventDetected" object:nil]; //消息的推送
        }else{
            [[NSNotificationCenter defaultCenter] postNotificationName:@"onMagicEventNotDetected" object:nil]; //消息的推送
        }
    }else{
        _lastTotalBrightnessValue = totalBrightness;
        [[NSNotificationCenter defaultCenter] postNotificationName:@"onMagicEventNotDetected" object:nil]; //消息的推送
    }
}
}
}

```

searchForBackCameraIfAvailable 方法实现对相机设备的获取。程序代码如下：

```

- (AVCaptureDevice *)searchForBackCameraIfAvailable
{
    NSArray *videoDevices = [AVCaptureDevice devicesWithMediaType:AVMediaTypeVideo];
    AVCaptureDevice *captureDevice = nil;
    //遍历
    for (AVCaptureDevice *device in videoDevices)

```

```

{
    //判断设备的位置是否为 AVCaptureDevicePositionBack
    if (device.position == AVCaptureDevicePositionBack)
    {
        captureDevice = device;
        break;
    }
}
//判断 captureDevice 是否为空
if (!captureDevice)
{
    captureDevice = [AVCaptureDevice defaultDeviceWithMediaType:
        AVMediaTypeVideo];
}
return captureDevice;
}

```

(7) 打开 ViewController.h 文件, 编写代码, 实现头文件、实例变量、对象、插座变量的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "Magic.h"
@interface ViewController : UIViewController
{
    BOOL _rabbitVisible; //布尔类型的实例变量
    Magic *magic;
    IBOutlet UIImageView *_imageView; //图像视图的插座变量
}
@end

```

(8) 打开 Main.storyboard 文件, 对 View Controller 视图控制器的设计界面进行设计, 效果如图 10.15 所示。

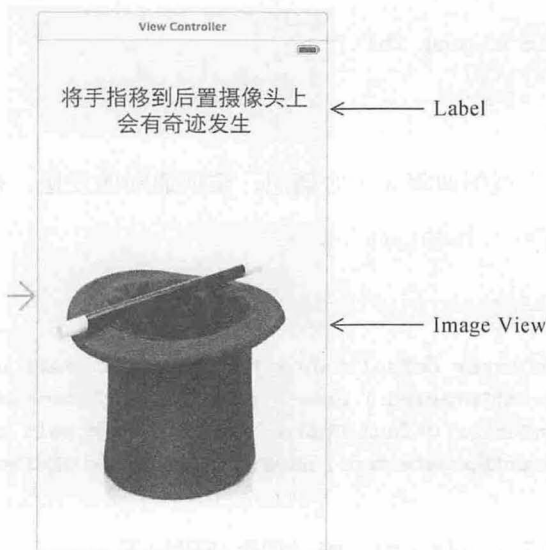


图 10.15 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 10-18 所示。

表 10-18 视图、控件设置

视图、控件	属 性 设 置	其 他
Label	Text: 将手指移到后置摄像头上有奇迹发生 Font: System 25.0 Alignment: 居中 Lines: 2	
Image View		与插座变量 scrollView 关联

(9) 打开 ViewController.m 文件, 编写代码, 实现在手遮挡住后置摄像头后出现小兔子的功能。使用的方法如表 10-19 所示。

表 10-19 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
viewWillAppear:	视图在即将显示时调用, 实现通知的发送
viewWillDisappear:	视图在即将消失时调用, 实现移除通知
receiveOnMagicEventDetected:	调用显示兔子方法
receiveOnMagicEventNotDetected:	调用隐藏兔子方法
showRabbit	显示兔子
hideRabbit	隐藏兔子

viewDidLoad 方法在视图加载后调用, 实现初始化的功能。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    self.view.backgroundColor = [UIColor whiteColor];
    //创建事件
    magic = [[Magic alloc] init];
    [magic startCapture];
    _rabbitVisible = NO;
}

```

viewWillAppear:方法在视图即将显示时调用, 实现通知的发送。程序代码如下:

```

- (void)viewWillAppear: (BOOL) animated
{
    [super viewWillAppear:animated];
    //发送通知
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector
(receiveOnMagicEventDetected:) name:@"onMagicEventDetected" object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector
(receiveOnMagicEventNotDetected:) name:@"onMagicEventNotDetected" object:nil];
}

```

showRabbit 方法实现显示小兔子的功能。程序代码如下:

```

- (void) showRabbit
{
    //判断
    if(!_rabbitVisible)

```

```

{
    _rabbitVisible = YES;
    _imageView.image = [UIImage imageNamed:@"2.jpg"];    //设置图像
}
}

```

【代码解析】

本实例关键功能是后置摄像头的获取以及兔子的出现和消失。下面依次讲解这两个知识点。

1. 后置摄像头的获取

在本实例中后置摄像头的获取首先需要创建一个捕获设备，然后再遍历数组中的捕获设备。在遍历期间，要对设备的位置是否为 `AVCaptureDevicePositionBack` 进行判断。如果是，就把当前的设备作为捕获设备，并返回此设备，从而实现后置摄像头的获取。程序代码如下：

```

NSArray *videoDevices = [AVCaptureDevice devicesWithMediaType:AVMediaTypeVideo];
AVCaptureDevice *captureDevice = nil;
for (AVCaptureDevice *device in videoDevices)
{
    if (device.position == AVCaptureDevicePositionBack)
    {
        captureDevice = device;
        break;
    }
}
if (!captureDevice)
{
    captureDevice = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];
}
return captureDevice;

```

2. 兔子的出现和消失

在本实例中兔子的出现和消失是通过对亮度级别的判断实现的。程序代码如下：

```

if([self
calculateLevelOfBrightness:totalBrightness]<_brightnessThreshold)
{
    if([self calculateLevelOfBrightness:totalBrightness]>MIN_BRIGHTNESS_THRESHOLD)
    {
        [[NSNotificationCenter defaultCenter] postNotificationName:
@"onMagicEventDetected" object:nil];
    }else{
        [[NSNotificationCenter defaultCenter] postNotificationName:
@"onMagicEventNotDetected" object:nil];
    }
}
else{
    _lastTotalBrightnessValue = totalBrightness;
    [[NSNotificationCenter defaultCenter] postNotificationName:
@"onMagicEventNotDetected" object:nil];
}
}

```

实例 113 录像机

【实例描述】

本实例实现的是一个录像机的功能。当用户单击“录像”按钮后，打开录像机实现录像视频的录制。在录制完成后，返回主界面。单击导航栏中的 play 按钮后，实现录制视频的播放，单击 stop 按钮后，停止录制视频的播放。运行效果如图 10.16 所示。



图 10.16 运行效果

【实现过程】

- (1) 创建一个项目，命名为“录像机”。
- (2) 添加 AssetsLibrary.framework、MobileCoreServices.framework 和 MediaPlayer.framework 到创建的项目中。
- (3) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、对象、插座变量以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
//头文件
#import <AssetsLibrary/AssetsLibrary.h>
#import <MobileCoreServices/MobileCoreServices.h>
#import <MediaPlayer/MediaPlayer.h>
@interface ViewController : UIViewController<UINavigationControllerDelegate,
UIImagePickerControllerDelegate>
{
    //对象
    UIImagePickerController *imagePicker;
    MPMoviePlayerController *movie;
    NSURL* mediaURL ;
    IBOutlet UIView *vv; //空白视图的插座变量
}
//动作
- (IBAction)show:(id)sender;
- (IBAction)play:(id)sender;
- (IBAction)stop:(id)sender;
@end
```

(4) 打开 Main.storyboard 文件，拖动 Navigation Controller 导航控制器到画布中。将此控制器关联的根视图设置为 View Controller 视图控制器的设计界面。对 View Controller 视图控制器的设计界面进行设计，效果如图 10.17 所示。

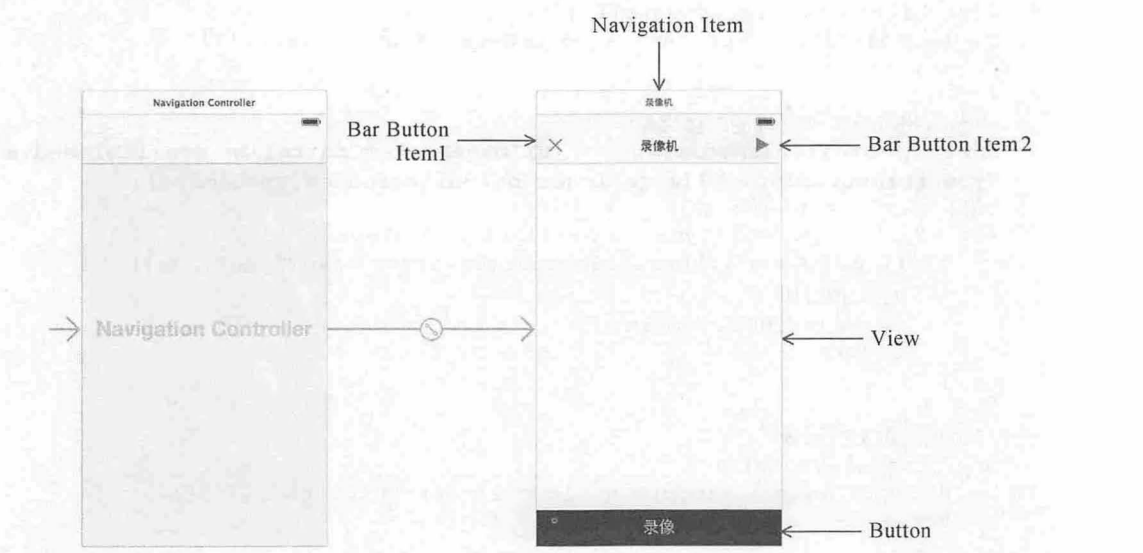


图 10.17 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 10-20 所示。

表 10-20 视图、控件设置

视图、控件	属 性 设 置	其 他
Navigation Item	Title: 录像机	
Bar Button Item1	Identifier: Stop	与动作 stop:关联
Bar Button Item2	Identifier: Play	与动作 play:关联
View		与插座变量 vv 关联
Button	Title: 录像 Font: System 21.0 Text Color: 白色 Background: 黑色	与动作 show:关联

(5) 打开 ViewController.m 文件，编写代码，实现录像机录像并播放的功能。使用的方法如表 10-21 所示。

表 10-21 ViewController.m文件中方法总结

方 法	功 能
show:	单击“相机”按钮，显示录像机
imagePickerController:didFinishPickingMediaWithInfo:	将录制的视频保存
imagePickerControllerDidCancel:	关闭相机应用
play:	播放录制的视频
stop:	结束播放的视频

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，show:方法实现

了在单击“相机”按钮后，显示录像机的功能。程序代码如下：

```
- (IBAction) show:(id) sender {
    //检查相机模式是否可用
    if (![UIImagePickerController isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera]) {
        NSLog(@"sorry, no camera or camera is unavailable!!!");
        return;
    }
    //获得相机模式下支持的媒体类型
    NSArray* availableMediaTypes = [UIImagePickerController availableMediaTypesForSourceType:UIImagePickerControllerSourceTypeCamera];
    BOOL canTakeVideo = NO;
    for (NSString* mediaType in availableMediaTypes) {
        if ([mediaType isEqualToString:(NSString *)kUTTypeImage]) {
            //支持摄像
            canTakeVideo = YES;
            break;
        }
    }
    //检查是否支持摄像
    if (!canTakeVideo) {
        NSLog(@"sorry, capturing video is not supported.!!!");
        return;
    }
    //创建图像选取控制器
    UIImagePickerController* imagePicker = [[UIImagePickerController alloc] init];
    //设置图像选取控制器的来源模式为相机模式
    imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;
    //设置图像选取控制器的类型为动态图像
    imagePicker.mediaTypes = [[NSArray alloc] initWithObjects:(NSString*)kUTTypeMovie, nil];
    //设置摄像图像品质
    imagePicker.videoQuality = UIImagePickerControllerQualityTypeHigh;
    //设置最长摄像时间
    imagePicker.videoMaximumDuration = 30;
    //允许用户进行编辑
    imagePicker.allowsEditing = YES;
    //设置委托对象
    imagePicker.delegate = self;
    //以模式视图控制器的形式显示
    [self presentViewController:imagePicker animated:YES completion:nil];
}
```

UIImagePickerController:didFinishPickingMediaWithInfo:方法实现将录制好的视频保存在相册中。程序代码如下：

```
- (void) UIImagePickerController:(UIImagePickerController *)picker didFinishPickingMediaWithInfo:(NSDictionary *)info{
    //获取视频文件的 url
    mediaURL = [info objectForKey:UIImagePickerControllerMediaURL];
    //创建 ALAssetsLibrary 对象并将视频保存到媒体库
    ALAssetsLibrary* assetsLibrary = [[ALAssetsLibrary alloc] init];
    //保存
    [assetsLibrary writeVideoAtPathToSavedPhotosAlbum:mediaURL completionBlock:^(NSURL *assetURL, NSError *error) {
        //判断是否错误
        if (!error) {
```



```

        NSLog(@"captured video saved with no error.");
    }else
    {
        NSLog(@"error occured while saving the video:%@", error);
    }
}];
[imagePicker dismissViewControllerAnimated:YES completion:nil];
}

```

play方法实现在单击 play 按钮后实现录制视频的播放。程序代码如下:

```

- (IBAction)play:(id)sender {
    movie=[MPMoviePlayerController alloc]initWithContentURL:mediaURL]; //创建
    [movie.view setFrame:vv.frame]; //设置框架
    [self.view addSubview:movie.view];
    movie.controlStyle=MPMovieControlStyleFullscreen;
    [movie play];
}

```

【代码解析】

本实例关键功能是录像机的显示以及录像视频的保存。下面就是这个知识点的详细讲解。

1. 录像机的显示

UIImagePickerController 的 mediaTypes 属性实现确定在图像选取控制器里显示哪些类型的多媒体文件,其语法形式如下:

```
@property (nonatomic, copy) NSArray *mediaTypes;
```

在本实例中就是将图像选取控制器显示的类型设置为视频类型,即 kUTTypeMovie。代码如下:

```

imagePicker.mediaTypes = [[NSArray alloc] initWithObjects:(NSString*)
kUTTypeMovie, nil];

```

2. 录像视频的保存

视频的保存一般使用 ALAssetsLibrary 的 writeVideoAtPathToSavedPhotosAlbum:completionBlock:方法。它的功能是将 URL 标识的视频文件保存在相册中。其语法形式如下:

```

- (void)writeVideoAtPathToSavedPhotosAlbum:(NSURL *)videoPathURL
completionBlock:
(ALAssetsLibraryWriteVideoCompletionBlock) completionBlock;

```

其中, (NSURL *)videoPathURL 表示 URL,它是指定的视频文件; (ALAssetsLibraryWriteVideoCompletionBlock) completionBlock 表示在保存操作完成后调用的块。在本实例中就是使用了 writeVideoAtPathToSavedPhotosAlbum:completionBlock:方法实现了对录制视频的保存功能。程序代码如下:

```

[assetsLibrary writeVideoAtPathToSavedPhotosAlbum:mediaURL completionBlock:
^(NSURL *assetURL, NSError *error) {
    if (!error) {
        NSLog(@"captured video saved with no error.");
    }else
    {
        NSLog(@"error occured while saving the video:%@", error);
    }
}];

```

第 11 章 传 感 器

在现在的手机中都内置了各种传感器，如加速计和陀螺仪传感器。这些传感器可以感知设备的方向、速度、加速度，所以在很多的应用中都使用了这些传感器，例如极品飞车、逃离神庙等。本章就通过这些传感器实现相应的一些实例。

实例 114 手机水平放置的测试

【实例描述】

本实例实现的功能是对手机是否水平放置做一个测试。当放置的手机不是水平时，图像就会呈现出一个相应的 3D 效果。运行效果如图 11.1 所示。



图 11.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“手机水平放置的测试”。
- (2) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 CoreMotion.framework 到创建的项目中。
- (4) 创建一个基于 UIImageView 类的分类 Tranform3D。
- (5) 打开 UIImageView+Tranform3D.h 文件，编写代码，实现方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface UIImageView (Tranform3D)
-(void)setRotation:(CGFloat)degress;
@end
```

(6) 打开 UIImageView+Tranform3D.m 文件, 编写代码, 实现对 3D 旋转效果的设置。程序代码如下:

```
-(void)setRotation:(CGFloat)degress
{
    CATransform3D transform = CATransform3DIdentity;
    transform.m34 = 1.0/100;
    CGFloat radiants = degress/360*M_PI;
    transform = CATransform3DRotate(transform, radiants, 1.0f, 0.0f, 0.0f); //旋转

    CALayer *layer = self.layer;
    layer.anchorPoint = CGPointMake(0.5, 0.5); //锚点
    layer.transform = transform;
}
```

(7) 创建一个基于 NSObject 类的 GetDegress 类。

(8) 打开 GetDegress.h 文件, 编写代码, 实现头文件、方法的声明。程序代码如下:

```
#import <Foundation/Foundation.h>
#import <CoreMotion/CoreMotion.h>
@interface GetDegress : NSObject
+ (void)getDegressWithBlock: (void (^)(CMAccelerometerData *latestAcc,
NSError *error))aBlock;
@end
```

(9) 打开 GetDegress.m 文件, 编写代码, 获取旋转角度。程序代码如下:

```
+ (void)getDegressWithBlock: (void (^)(CMAccelerometerData *latestAcc,
NSError *error))aBlock
{
    CMMotionManager *motionManager = [[CMMotionManager alloc] init];
    //判断加速计是否可用
    if (!motionManager.accelerometerAvailable) {
        NSLog(@"没有加速计");
    }
    motionManager.accelerometerUpdateInterval = 0.1; //更新频率是 100Hz
    [motionManager startDeviceMotionUpdates]; //开始更新
    [motionManager startAccelerometerUpdatesToQueue:[NSOperationQueue
currentQueue] withHandler:^(CMAccelerometerData *latestAcc, NSError
*error)
    {
        double a = motionManager.deviceMotion.gravity.x; //获取 x 的值
        aBlock(latestAcc, error);
    }];
}
```

(10) 打开 ViewController.h 文件, 编写代码, 实现头文件、插座变量的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
//头文件
#import "GetDegress.h"
#import "UIImageView+Tranform3D.h"
@interface ViewController : UIViewController{
    IBOutlet UIImageView *imageView; //图像视图的插座变量
}
@end
```

(11) 打开 Main.storyboard 文件，从视图库中拖动 Image View 图像视图到设计界面中，将此视图的 Image 属性设置为 1.png。

(12) 打开 ViewController.m 文件，编写代码，通过手机的不同放置而旋转图像。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [GetDegress getDegressWithBlock:^(CMAccelerometerData *latestAcc,
    NSError *error) {
        //判断 y 是否小于 0 大于等于 -1.0
        if (latestAcc.acceleration.y < 0.0 && latestAcc.acceleration.y >= -1.0) {
            [imageView setRotation:80*latestAcc.acceleration.y]; //设置旋转
        }
        else if (latestAcc.acceleration.z*-1 > 0.0 && latestAcc.acceleration.
        z*-1 <= 1.0)
        {
            [imageView setRotation:80 - ( 80*latestAcc.acceleration.z*-1)];
            //设置旋转
            NSLog(@"-----%f",80*latestAcc.acceleration.z*-1);
        }
    }];
}
```

【代码解析】

本实例关键功能是手机的水平测试。下面就是这个知识点的详细讲解。

加速计是一种可以感应在设备中一个方向上的线性加速度的传感器。在本实例中对于水平的测试就是使用了加速计传感器实现。在使用加速计传感器之前，首先需要判断当前设备的传感器是否可用，这时需要使用 accelerometerAvailable 属性实现。其语法形式如下：

```
@property(readonly, nonatomic, getter=isAccelerometerAvailable) BOOL
accelerometerAvailable
```

其中，属性值为 YES 或者为 1 时，表示加速度传感器可用，如果为 NO 或者为 0 时，表示加速度传感器不可用。在本实例中就使用了 AccelerometerAvailable 方法对加速计传感器是否可用做出了判断。代码如下：

```
if (!motionManager.accelerometerAvailable) {
    NSLog(@"没有加速计");
}
```

在本实例中图像的 3D 效果是通过加速计数据采集实现的，其中需要使用 startAccelerometerUpdatesToQueue:withHandler:方法。此方法主要是使用操作队列数据进行采样。其语法形式如下：

```
- (void)startAccelerometerUpdatesToQueue:(NSOperationQueue *)queue withHandler:
(CMAccelerometerHandler)handler;
```

其中，(NSOperationQueue *)queue 表示操作队列；(CMAccelerometerHandler)handler 表示在每一次更新处理新的加速度数据时调用的块方法。在本实例中的代码如下：

```
[motionManager startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
withHandler:^(CMAccelerometerData *latestAcc, NSError *error)
{
    double a = motionManager.deviceMotion.gravity.x;
    aBlock(latestAcc,error);
}];
```

实例 115 加速的小球

【实例描述】

本实例实现了加速度传感器实现了一个加速的小球。在调整设备后小球就会做加速运动；当单击“结束”按钮，小球的加速运动也会停止。运行效果如图 11.2 所示。



图 11.2 运行效果

【实现过程】

- (1) 创建一个项目，命名为“加速的小球”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 CoreMotion.framework 到创建的项目中。
- (4) 创建一个基于 UIView 类的 BallView 类。
- (5) 打开 BallView.h 文件，编写代码，实现头文件、协议、属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import <CoreMotion/CoreMotion.h>
//协议
@protocol BallViewDelegate
- (void)updateLabelWithX:(double) accelerometerX Y:(double) accelerometerY;
@end
@interface BallView : UIView
//属性
@property (readwrite) float velocity;
@property (nonatomic, strong) CMMotionManager *mManager;
@property (nonatomic, weak) id<BallViewDelegate> delegate;
```



```
//方法
- (void)startUpdateAccelerometer;
- (void)stopUpdate;
@end
```

(6) 打开 BallView.m 文件，编写代码，实现小球加速的效果，使用的方法如表 11-1 所示。

表 11-1 BallView.m文件中方法总结

方 法	功 能
initWithFrame:	初始化
mManager	获取 CMMotionManager 对象
startUpdateAccelerometer	开始更新加速计
stopUpdate	停止更新

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，startUpdateAccelerometer 方法实现开始更新加速计的功能。程序代码如下：

```
- (void)startUpdateAccelerometer
{
    //设置采样的频率
    NSTimeInterval updateInterval = 0.07;
    CGSize size = [self superview].frame.size;
    __block CGRect f = [self frame];
    //在 block 中，只能使用 weakSelf。
    BallView * __weak weakSelf = self;
    //判断是否加速度传感器可用，如果可用则继续
    if ([self.mManager isAccelerometerAvailable] == YES) {
        //给采样频率赋值
        //加速度传感器开始采样
        [self.mManager startAccelerometerUpdatesToQueue:[NSOperationQueue
currentQueue] withHandler:^(CMAccelerometerData *accelerometerData,
NSError *error)
        {
            f.origin.x += (accelerometerData.acceleration.x * weakSelf.
velocity) * 1;
            f.origin.y += (accelerometerData.acceleration.y * weakSelf.
velocity) * -1; if(f.origin.x < 0)           //判断 f 区域的 x 是否小于 0
                f.origin.x = 0;
            if(f.origin.y < 0)                 //判断 f 区域的 y 是否小于 0
                f.origin.y = 0;
            if(f.origin.x > (size.width - f.size.width))
                f.origin.x = (size.width - f.size.width); //设置 f 区域的 x 值
            if(f.origin.y > (size.height - f.size.height))
                f.origin.y = (size.height - f.size.height); //设置 f 区域的 y 值
            //运动动画
            [UIView beginAnimations:nil context:nil];
            [UIView setAnimationDuration:0.1];
            [weakSelf setFrame:f];              //设置框架
            [UIView commitAnimations];
            [weakSelf.delegate updateLabelWithX:accelerometerData.
acceleration.x Y:accelerometerData.acceleration.y];
        }];
    }
}
```

(7) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、插座变量、对象以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "BallView.h"
@interface ViewController : UIViewController<BallViewDelegate>
{
    //插座变量
    IBOutlet UIView *boulderView;
    IBOutlet UILabel *label;
    IBOutlet UIButton *button;
    //对象
    BallView *ballView;
    UIImageView *imageView;
}
- (IBAction)Clicked:(id)sender;
@end
```

(8) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 11.3 所示。

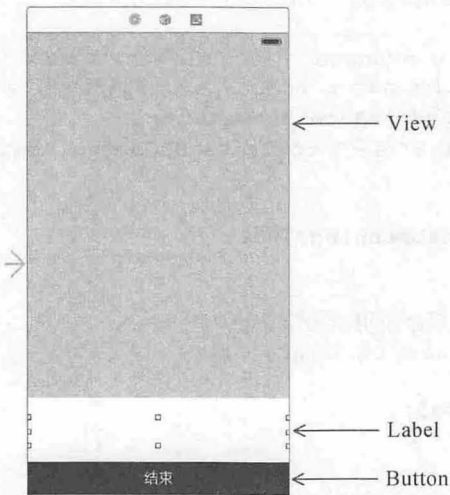


图 11.3 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 11-2 所示。

表 11-2 视图、控件设置

视图、控件	属 性 设 置	其 他
View	Title: 应用	与插座变量 boulderView 关联
Label	Text: (空) Font: System 21.0 Alignment: 居中	与插座变量 label 关联
Button	Title: 结束 Font: System 18.0 Text Color: 白色 Background: 黑色	与插座变量 button 关联 与动作 Clicked:关联

(9) 打开 ViewController.m 文件，编写代码，实现加速的小球。程序代码如下：

```
//视图加载后调用，实现初始化
- (void) viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    CGRect gravityBallViewFrame = CGRectMake(0, 0, 40, 40);
    ballView = [[BallView alloc] initWithFrame:gravityBallViewFrame];
    [boudnerView addSubview:ballView]; //添加视图对象
    ballView.delegate = self; //设置委托
    //创建并设置图像视图
    imageView = [[UIImageView alloc] initWithFrame:gravityBallViewFrame];
    imageView.image = [UIImage imageNamed:@"1.png"]; //设置图像
    [ballView addSubview:imageView];
    [ballView startUpdateAccelerometer]; //开始更新加速计
}

//单击按钮后的响应
- (IBAction) Clicked: (id) sender {
    if ([ballView.mManager isAccelerometerActive] == YES){
        [ballView stopUpdate]; //停止更新
        [button setTitle:@"开始" forState:UIControlStateNormal]; //设置按钮的标题
    } else if ([ballView.mManager isAccelerometerActive] != YES) &&
        ([ballView.mManager isAccelerometerAvailable] == YES){
        [ballView startUpdateAccelerometer]; //开始更新加速计
        [button setTitle:@"结束" forState:UIControlStateNormal]; //设置按钮的标题
    } else{
        NSLog(@"what's happenning?\n");
    }
}

//在视图即将显示时调用，实现结束更新加速计的功能
- (void) viewWillAppear: (BOOL) animated
{
    [ballView stopUpdate];
}

//在标签中显示更新的内容
- (void) updateLabelWithX: (double) accelerometerX Y: (double) accelerometerY
{
    label.text = [NSString stringWithFormat:@"X:%f-Y:%f", accelerometerX,
        accelerometerY];
}
```

【代码解析】

本实例关键功能是小球的加速运动。下面就是这个知识点的详细讲解。

对于小球的加速运行效果，需要使用 CMMotionManager 的 startAccelerometerUpdatesToQueue:withHandler:方法实现，此方法主要是使用操作队列的数据进行采样。代码如下：

```
[self.mManager startAccelerometerUpdatesToQueue:[NSOperationQueue currentQueue]
withHandler:^(CMAccelerometerData *accelerometerData, NSError *error){
    .....
}];
```

实例 116 摇一摇音乐播放器

【实例描述】

在很多的音乐播放器中都采用了摇一摇就可以切换播放的音乐这一功能,使用户摆脱了在界面进行切换音乐的命运。本实例就来实现摇一摇切换音乐的功能。运行效果如图 11.4 所示。



图 11.4 运行效果

【实现过程】

- (1) 创建一个项目, 命名为“摇一摇音乐播放器”。
- (2) 添加图像 1.jpg、2.png、3.png 到创建项目的 Supporting Files 文件夹中。
- (3) 添加音频文件“孤单北半球.mp3”、“天使的翅膀.mp3”、“狐狸雨.mp3”、“一样爱着你.mp3”到创建项目的 Supporting Files 文件夹中。
- (4) 添加 AVFoundation.framework 到创建的项目中。
- (5) 打开 ViewController.m 文件, 编写代码, 实现头文件、遵守协议、插座变量、对象、实例变量以及动作的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
@interface ViewController : UIViewController<AVAudioPlayerDelegate>
{
    //插座变量
    IBOutlet UILabel *label;
    IBOutlet UIButton *button;
    //对象
    AVAudioPlayer* audioPlayer;
    NSMutableArray* pNames;
    NSMutableArray* hNames;
    //实例变量
    int _songIndex;
    BOOL isPlay;
}
```

```
- (IBAction)Clicked:(id) sender;
@end
```

（6）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 11.5 所示。

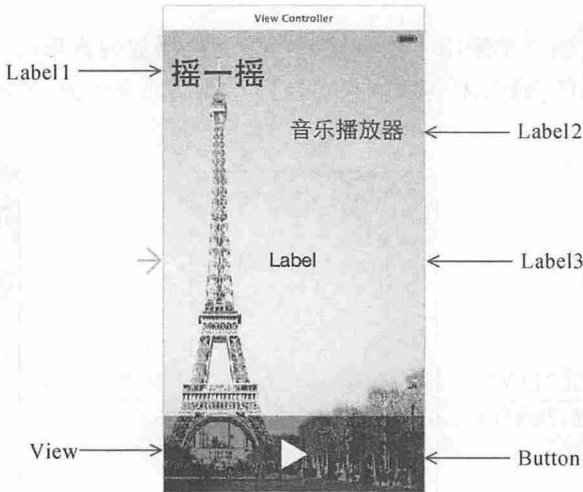


图 11.5 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 11-3 所示。

表 11-3 视图、控件设置

视图、控件	属 性 设 置	其 他
Label1	Text: 摇一摇 Font: System Bold 39.0 Alignment: 居中	
Label2	Text: 音乐播放器 Font: System 28.0 Alignment: 居中	
Label3	Font: System 24.0 Alignment: 居中	与插座变量 label 关联
View	Alpha: 0.4 Background: 黑色	
Button	Background: 2.png	与插座变量 button 关联 与动作 Clicked:关联

（7）打开 ViewController.m 文件，编写代码，实现播放音乐以及摇一摇更换音乐的功能。使用的方法如表 11-4 所示。

表 11-4 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
dataInit	初始化音频数据
loadMusic:type:	加载音乐

续表

方 法	功 能
Clicked:	单击播放按钮, 实现播放或暂停
audioPlayerDidFinishPlaying:successfully:	在播放完成后调用, 实现自动播放
motionEnded:withEvent:	摇一摇

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, viewDidLoad 方法在实现加载后调用, 实现初始化的功能。程序代码如下:

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    [self performSelector:@selector(dataInit)];
    [self loadMusic:[pNames objectAtIndex:_songIndex] type:@"mp3"];
    //加载音乐
    label.text= [hNames objectAtIndex:_songIndex];    //设置文本内容
}
```

loadMusic:type:方法实现音乐的加载。程序代码如下:

```
-(void)loadMusic:(NSString*)name type:(NSString*)type
{
    NSString* path= [[NSBundle mainBundle] pathForResource: name ofType:type];
    NSURL* url = [NSURL fileURLWithPath:path];
    audioPlayer= [[AVAudioPlayer alloc] initWithContentsOfURL:url error:nil];
    audioPlayer.delegate=self;    //设置委托
    audioPlayer.volume= 0.5;    //设置音量
    [audioPlayer prepareToPlay];
}
```

Clicked:方法实现单击“播放”按钮后, 音乐开始播放或暂停。程序代码如下:

```
-(IBAction)Clicked:(id)sender {
    //判断 isPlayer 是否为 YES
    if (isPlay) {
        [audioPlayer play];    //开始播放音乐
        [button setBackgroundImage:[UIImage imageNamed:@"3.png"] forState:
        UIControlStateNormal];
        isPlay = NO;
    } else {
        [audioPlayer pause];    //暂停播放的音乐
        [button setBackgroundImage:[UIImage imageNamed:@"2.png"] forState:
        UIControlStateNormal];
        isPlay = YES;
    }
}
```

audioPlayerDidFinishPlaying:方法实现在播放完一曲音乐后, 自动播放下一曲音乐的功能。程序代码如下:

```
-(void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player successfully:
(BOOL)flag
{
    _songIndex++;
    if (_songIndex==pNames.count)
        _songIndex= 0;
}
```

```
[self loadMusic:[pNames objectAtIndex:_songIndex] type:@"mp3"];
//加载音乐
label.text= [hNames objectAtIndex:_songIndex];
//设置文本
[audioPlayer play];
//播放音乐
}
```

motionEnded:withEvent:方法实现摇一摇的功能，即摇一摇后更换音乐的功能。程序代码如下：

```
- (void) motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event {
//判断 motion 是否为 UIEventSubtypeMotionShake
if (motion == UIEventSubtypeMotionShake) {
    if (_songIndex== hNames.count - 1) {
        _songIndex = -1;
    }
    _songIndex ++;
    [self updatePlayerSetting];
//更新音乐的设置
}
}
```

updatePlayerSetting 方法实现对音乐功能的设置。程序代码如下：

```
- (void)updatePlayerSetting {
    [self loadMusic:[pNames objectAtIndex:_songIndex] type:@"mp3"];
    label.text= [hNames objectAtIndex:_songIndex];
//设置文本内容
    [audioPlayer play];
//播放音乐
}
```

【代码解析】

本实例关键功能是摇一摇更换音乐的功能。下面就是这个知识点的详细讲解。

UIResponder 的 motionBegan:withEvent:、motionCancelled:withEvent:、motionEnded:withEvent:方法实现运动事件，其类似于触摸事件。motionBegan:withEvent:方法在开始运动时调用，其语法形式如下：

```
- (void)motionBegan:(UIEventSubtype)motion withEvent:(UIEvent *)event;
```

motionCancelled:withEvent:方法在取消运动时调用，其语法形式如下：

```
- (void)motionCancelled:(UIEventSubtype)motion withEvent:(UIEvent *)event;
```

motionEnded:withEvent:方法在运动结束时调用，其语法形式如下

```
- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event;
```

其中，(UIEventSubtype)motion 是一个事件类型常量，表示与运动关联的事件。这些常量如表 11-5 所示。

表 11-5 常量

事件类型常量	说 明
UIEventSubtypeNone	表示触摸事件
UIEventSubtypeMotionShake	表示摇动事件
UIEventSubtypeRemoteControlPlay	表示多媒体远程控制事件
UIEventSubtypeRemoteControlPause	
UIEventSubtypeRemoteControlStop	

续表

事件类型常量	说 明
UIEventSubtypeRemoteControlTogglePlayPause	表示多媒体远程控制事件
UIEventSubtypeRemoteControlNextTrack	
UIEventSubtypeRemoteControlPreviousTrack	
UIEventSubtypeRemoteControlBeginSeekingBackward	
UIEventSubtypeRemoteControlEndSeekingBackward	
UIEventSubtypeRemoteControlBeginSeekingForward	
UIEventSubtypeRemoteControlEndSeekingForward	

在本实例中就是使用了 `motionEnded:withEvent:` 方法实现实现了摇一摇更换音乐的功能。程序代码如下：

```

- (void) motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event {
    if (motion == UIEventSubtypeMotionShake) {
        .....
    }
}

```

实例 117 根据手机转动显示图像

【实例描述】

本实例的功能是通过陀螺仪实现旋转手机显示图像的部分。当用户左旋转手机时，滚动视图中的图像就会向左滚动；用户右旋转手机时，滚动视图中的图像就会向右滚动。运行效果如图 11.6 所示。

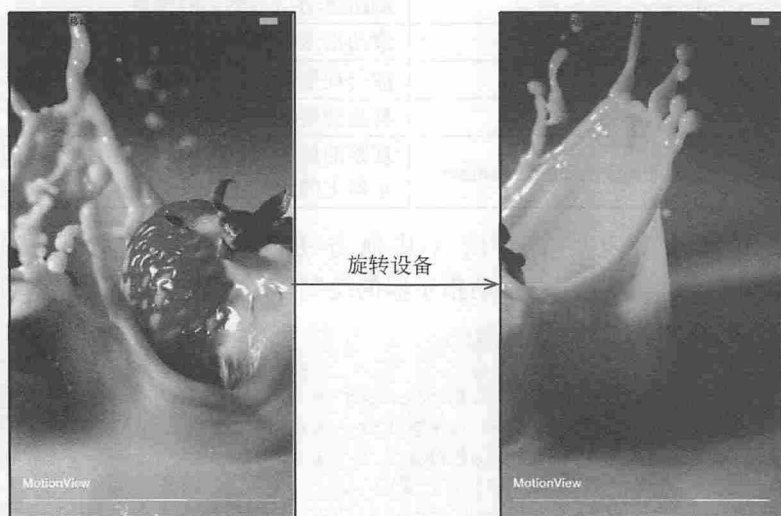


图 11.6 运行效果

【实现过程】

(1) 创建一个项目，命名为“根据手机转动显示图像”。

- (2) 添加图像 1.jpg 到创建项目的 Supporting Files 文件夹中。
- (3) 添加 CoreMotion.framework 到创建的项目中。
- (4) 创建一个基于 UIScrollView 类的分类 ScrollIndicator。
- (5) 打开 UIScrollView+ScrollIndicator.h 文件，编写代码，实现头文件、方法的声明。

程序代码如下：

```
#import <UIKit/UIKit.h>
#import <objc/runtime.h>
@interface UIScrollView (ScrollIndicator)
- (void)cr_enableScrollIndicator;           //滚动指示器
- (void)cr_disableScrollIndicator;         //滚动指示器禁用
- (void)cr_refreshScrollIndicator;         //刷新滚动指示器
@end
```

(6) 打开 UIScrollView+ScrollIndicator.m 文件，编写代码，实现滚动指示器的功能。使用的方法如表 11-6 所示。

表 11-6 UIScrollView+ScrollIndicator.m 文件中方法总结

方 法	功 能
cr_setViewForScrollIndicator:	设置滚动指示器的视图，实现一个对象和另一个对象的关联
cr_getViewForScrollIndicator	获取滚动指示器的视图
cr_setBackgroundViewForScrollIndicator:	设置滚动指示器背景视图，实现一个对象和另一个对象的关联
cr_getBackgroundViewForScrollIndicator	获取背景视图
cr_enableScrollIndicator	滚动指示器
cr_refreshScrollIndicator	刷新滚动指示器
cr_refreshBackgroundViewScrollIndicator	刷新滚动指示器的背景视图
cr_refreshScrollViewIndicator	刷新滚动指示器上的滑块
cr_disableScrollIndicator	滚动指示器禁用
cr_setupObservers	注册观察者
cr_unsetObservers	移除观察者
observeValueForKeyPath:ofObject:change:context:	观察的属性变化时会自动调用，即实现刷新滚动指示器上的滑块

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，cr_enableScrollIndicator 方法实现对滚动指示器的绘制以及滚动。程序代码如下：

```
- (void)cr_enableScrollIndicator
{
    self.showsHorizontalScrollIndicator = NO;           //不显示水平滚动指示器
    UIColor *indicatorColor = [UIColor whiteColor];    //创建一个白色的颜色对象
    CGFloat backgroundIndicatorWidth = self.frame.size.width - (CRScrollIndicatorLeftRightThreshold * 2);
    CGRect backgroundIndicatorFrame = CGRectMake(self.contentOffset.x +
        (self.frame.size.width / 2) - (backgroundIndicatorWidth / 2), self.frame.size.height -
        CRScrollIndicatorHeight - CRScrollIndicatorBottomSpace, backgroundIndicatorWidth,
        CRScrollIndicatorHeight);
    //创建空白视图，作为指示器的背景
    UIView *backgroundViewScrollIndicator = [[UIView alloc] initWithFrame:
        backgroundIndicatorFrame];
```

```

//设置背景颜色
[backgroundViewScrollIndicator setBackgroundColor:[indicatorColor
 colorWithAlphaComponent:0.50]];
[self cr_setBackgroundViewForScrollIndicator:backgroundViewScroll
 Indicator];
[self addSubview:backgroundViewScrollIndicator];
//创建并设置空白视图作为指示器的滑块
CGFloat viewScrollIndicatorWidth = (self.bounds.size.width - (CScrollView
 IndicatorLeftRightThreshold * 2)) * (self.bounds.size.width - (CScrollView
 IndicatorLeftRightThreshold * 2)) / self.contentSize.width;
//判断 viewScrollIndicatorWidth 是否小于 CScrollViewIndicatorDefaultWidth
if (viewScrollIndicatorWidth < CScrollViewIndicatorDefaultWidth) {
    viewScrollIndicatorWidth = CScrollViewIndicatorDefaultWidth;
}
CGRect frame = CGRectMake(self.contentOffset.x + (self.frame.size.width
 / 2) - (viewScrollIndicatorWidth / 2), self.frame.size.height - CScrollView
 IndicatorHeight - CScrollViewIndicatorBottomSpace, viewScrollIndicatorWidth,
 CScrollViewIndicatorHeight); //创建矩形区域
UIView *viewScrollIndicator = [[UIView alloc] initWithFrame:frame];
[viewScrollIndicator setBackgroundColor:[indicatorColor colorWith
 AlphaComponent:1.0f]]; //设置背景颜色
[self cr_setViewForScrollIndicator:viewScrollIndicator];
[self addSubview:viewScrollIndicator]; //添加视图对象
[self cr_setupObservers];
}

```

`cr_refreshBackgroundViewScrollIndicator` 方法实现滚动指示器背景视图的刷新。程序代码如下:

```

- (void)cr_refreshBackgroundViewScrollIndicator
{
    UIView *backgroundViewScrollIndicator = [self cr_getBackgroundView
 ForScrollIndicator];
    CGRect rect = backgroundViewScrollIndicator.frame; //获取框架
    CGFloat x = self.contentOffset.x + CScrollViewIndicatorLeftRightThreshold;
    rect.origin.x = x;
    backgroundViewScrollIndicator.frame = rect; //设置框架
}

```

`cr_refreshScrollViewIndicator` 方法实现刷新滚动指示器上的滑块。程序代码如下:

```

- (void)cr_refreshScrollViewIndicator
{
    UIView *viewScrollIndicator = [self cr_getViewForScrollIndicator];
    CGRect rect = viewScrollIndicator.frame; //获取框架
    CGFloat percent = self.contentOffset.x / self.contentSize.width;
    CGFloat x = self.contentOffset.x + ((self.bounds.size.width - CScrollView
 IndicatorLeftRightThreshold) * percent) + CScrollViewIndicatorLeftRight
 Threshold;
    rect.origin.x = x;
    viewScrollIndicator.frame = rect; //设置框架
}

```

(7) 创建一个基于 `UIView` 类的 `MotionView` 类。

(8) 打开 `Motion.h` 文件, 编写代码, 实现头文件、属性以及方法的声明。程序代码如下:

```
#import <UIKit/UIKit.h>
```



```
//头文件
#import <CoreMotion/CoreMotion.h>
#import "UIScrollView+ScrollIndicator.h"
@interface MotionView : UIView
//属性
@property (nonatomic, strong) UIImage *image;
@property (nonatomic, assign, getter = isMotionEnabled) BOOL motionEnabled;
.....
- (instancetype)initWithFrame:(CGRect)frame image:(UIImage *)image;
@end
```

(9) 打开 Motion.m 文件，编写代码，实现运动视图的功能，即旋转设备使图像移动显示，使用的方法如表 11-7 所示。

表 11-7 Motion.m文件中方法总结

方 法	功 能
initWithFrame:	使用框架进行初始化
initWithFrame:image:	使用框架以及图像进行初始化
commonInit	初始化默认设置
setImage:	设置图像
setMotionEnabled:	设置运动是否启动
startMonitoring	开始陀螺仪的更新
stopMonitoring	结束更新

其中，commonInit 方法实现初始化默认设置。程序代码如下：

```
- (void)commonInit
{
    //创建并设置滚动视图
    _scrollView = [[UIScrollView alloc] initWithFrame:_viewFrame];
    [_scrollView setUserInteractionEnabled:NO];
    [_scrollView setBounces:NO];
    [_scrollView setContentSize:CGSizeZero];          //设置可滚动区域的尺寸
    [self addSubview:_scrollView];
    //创建并设置图像视图
    _imageView = [[UIImageView alloc] initWithFrame:_viewFrame];
    [_imageView setBackgroundColor:[UIColor blackColor]]; //设置背景颜色
    [_scrollView addSubview:_imageView];
    _minimumXOffset = 0;
    [self startMonitoring];
}
```

setImage:方法实现对图像的设置。程序代码如下：

```
- (void)setImage:(UIImage *)image
{
    _image = image;
    CGFloat width = _viewFrame.size.height / _image.size.height * _image.size.width;
    //对图像视图对象的设置
    [_imageView setFrame:CGRectMake(0, 0, width, _viewFrame.size.height)];
    [_imageView setBackgroundColor:[UIColor blackColor]]; //设置背景颜色
    [_imageView setImage:_image];                          //设置图像
    //对滚动视图对象的设置
```

```

_scrollView.contentSize = CGSizeMake(_imageView.frame.size.width,
_scrollView.frame.size.height);
_scrollView.contentOffset = CGPointMake((_scrollView.contentSize.
width - _scrollView.frame.size.width) / 2, 0); //设置可滚动区域的偏移量
[_scrollView cr_enableScrollIndicator];
_motionRate = _image.size.width / _viewFrame.size.width * CRMotionView
RotationFactor;
_maximumXOffset = _scrollView.contentSize.width - _scrollView.frame.
size.width;
}

```

startMonitoring 方法实现开始对陀螺仪进行更新的功能。程序代码如下:

```

- (void)startMonitoring
{
    //判断_motionManager 是否为空
    if (!_motionManager) {
        _motionManager = [[CMMotionManager alloc] init];
        _motionManager.gyroUpdateInterval = CRMotionGyroUpdateInterval;
    }
    //判断陀螺仪是否可用
    if (![ _motionManager isGyroActive] && [ _motionManager isGyroAvailable]) {
        //陀螺仪传感器开始采样
        [_motionManager startGyroUpdatesToQueue:[NSOperationQueue currentQueue]
        withHandler:^(CMGyroData *gyroData, NSError *error) {
            CGFloat rotationRate = gyroData.rotationRate.y; //获取 y 值
            if (fabs(rotationRate) >= CRMotionViewRotationMinimumTreshold) {
                CGFloat offsetX = _scrollView.contentOffset.x - rotationRate
                * _motionRate;
                if (offsetX > _maximumXOffset) {
                    offsetX = _maximumXOffset; //设置 offsetX
                } else if (offsetX < _minimumXOffset) {
                    offsetX = _minimumXOffset; //设置 offsetX
                }
                //动画效果
                [UIView animateWithDuration:0.3f delay:0.0f options:UIView
                AnimationOptionBeginFromCurrentState | UIViewAnimationOption
                AllowUserInteraction | UIViewAnimationOptionCurveEaseOut
                animations:^(
                    //设置可滚动区域的偏移量
                    [_scrollView setContentOffset:CGPointMake(offsetX, 0)
                    animated:NO];
                )completion:nil];
            }
        }];
    } else {
        NSLog(@"There is not available gyro.");
    }
}

```

(10) 打开 ViewController.h 文件, 编写代码, 实现头文件、对象的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "MotionView.h"
@interface ViewController : UIViewController

```

```

{
    //对象
    MotionEvent *motionView;
    UILabel *titleLabel;
}
@end

```

(11) 打开 ViewController.m 文件，编写代码，实现运动视图的显示。程序代码如下：

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //创建并设置运动视图对象
    motionView = [[MotionView alloc] initWithFrame:self.view.bounds];
    [motionView setImage:[UIImage imageNamed:@"1.jpg"]]; //设置图像
    [self.view addSubview:motionView];
    //创建并设置标签对象
    titleLabel = [[UILabel alloc] initWithFrame:CGRectMake(16, self.view.
frame.size.height - 50, 110, 20)];
    [titleLabel setText:@"MotionView"];
    [titleLabel setShadowOffset:CGSizeMake(0, 1.0f)]; //设置阴影颜色
    [titleLabel setShadowColor:[UIColor blackColor] colorWithAlpha
Component:0.2f];
    [titleLabel setTextColor:[UIColor whiteColor]];
    [titleLabel setFont:[UIFont boldSystemFontOfSize:14.0f]]; //设置文本
    [motionView addSubview:titleLabel];
}

```

【代码解析】

本实例关键功能通过手机旋转显示图像。下面就是这个知识点的详细讲解。

用来检测设备的旋转、横滚等功能，都是使用陀螺仪传感器实现的。在本实例中，手机旋转显示图像的功能，就是通过陀螺仪传感器对手机旋转的检测来实现的。在使用陀螺仪之前，首先需要对陀螺仪传感器的可用性进行判断，就和加速度传感器是一样的，在一些设备中是不支持的，例如模拟器中。它的判断需要使用 GyroAvailable 方法。

```
@property(readonly, nonatomic, getter=isGyroAvailable) BOOL gyroAvailable;
```

其中，如果属性值为 YES 或者为 1 时，表示加速度传感器可用；如果为 NO 或者为 0 时，表示加速度传感器不可用。在本实例中就使用了 gyroAvailable 方法对陀螺仪传感器是否可用做出了判断。代码如下：

```

if (!_motionManager.gyroActive && _motionManager.gyroAvailable) {
    .....
} else {
    .....
}

```

在本实例中图像的部分显示，需要使用 startGyroUpdatesToQueue:withHandler:方法。此方法主要是使用操作队列数据进行采样。其语法形式如下：

```

- (void)startGyroUpdatesToQueue:(NSOperationQueue *)queue withHandler:
(CMGyroHandler) handler

```

其中, (NSOperationQueue *)queue 表示操作队列; (CMAccelerometerHandler)handler 表示在每一次更新处理新的加速度数据时调用的块方法。在本实例中的代码如下:

```
[_motionManager startGyroUpdatesToQueue:[NSOperationQueue currentQueue]
withHandler:^(CMGyroData *gyroData, NSError *error) {
    .....
    [UIView animateWithDuration:0.3f delay:0.0f options:UIViewAnimationOptionBeginFromCurrentState |
    UIViewAnimationOptionAllowUserInteraction |
    UIViewAnimationOptionCurveEaseOut animations:^(
        [_scrollView setContentOffset:CGPointMake(offsetX, 0) animated:NO];
    )completion:nil];
    .....
}];
```

第 12 章 网 络

网络是与外部世界通信的一种手段，在 iOS 中使用网络可以增强应用程序代码的趣味性和实用性。本实例将为读者实现一些与网络有关的实例。例如，利用 XML 解析可以获取手机的电话号码，利用第三方库 AFNetworking 可以实现图像下载队列控制等。

实例 118 手机号码查询

【实例描述】

在很多的应用中都安装了手机号码查询的功能，例如 360 手机卫士。本实例就为各位读者实现此功能。在文本框中输入手机号码后，单击“查询”按钮，就会弹出一个具有手机号码信息的警告视图。在此视图中详细介绍了手机的归属地以及手机卡的类型。运行效果如图 12.1 所示。

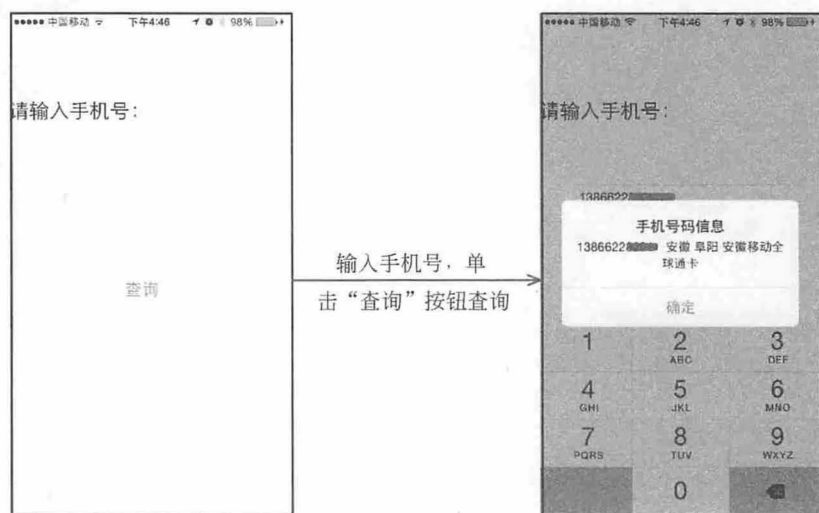


图 12.1 运行效果

【实现过程】

- (1) 创建一个项目，命名为“手机号码查询”。
- (2) 创建一个基于 NSObject 类的 SoapNAL 类。
- (3) 打开 SoapNAL.h 文件，编写代码，实现宏定义、类型定义、遵守协议、对象、实例变量、属性以及方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
```



```

#define MacthingElement @"getMobileCodeInfoResult"
typedef void (^SoapNALBlock) (NSMutableString *parserXML);
@interface SoapNAL : NSObject<NSXMLParserDelegate>{
    NSXMLParser *xmlParser;
    BOOL elementFound;
}
//属性
@property(nonatomic,strong) SoapNALBlock soapBlock;
@property(nonatomic,strong) NSMutableString *soapResults;
//方法
+(SoapNAL *)shareInstance;
-(void)parserSoapXML:(NSMutableData *)soapData withParserBlock:(SoapNALBlock)
block;
@end

```

(4) 打开 SoapNAL.m 文件, 编写代码, 实现电话号码的解析功能。使用的方法如表 12-1 所示。

表 12-1 SoapNAL.h文件中方法总结

方 法	功 能
shareInstance	获取单例
parserSoapXML:withParserBlock:	解析 XML
parser:didStartElement:namespaceURI:qualifiedName:attributes:	开始解析一个元素名
parser:foundCharacters:	追加找到的元素值
parser:didEndElement:namespaceURI:qualifiedName:	找到某一元素结尾处调用, 结束解析这个元素名
parserDidEndDocument:	解析结束后调用
parser:parseErrorOccurred:	出错时调用

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, parserSoapXML:withParserBlock:方法实现对 XML 的解析。程序代码如下:

```

-(void)parserSoapXML:(NSMutableData *)soapData withParserBlock:(SoapNALBlock)
block{
    self.soapBlock=block;
    xmlParser=[[NSXMLParser alloc] initWithData:soapData];
    xmlParser.delegate=self;
    [xmlParser parse];           //开始解析
}

```

parser:didStartElement:namespaceURI:qualifiedName:attributes:方法实现开始解析一个元素名。程序代码如下:

```

-(void) parser:(NSXMLParser *) parser didStartElement:(NSString *)
elementName namespaceURI:(NSString *) namespaceURI qualifiedName:(NSString
*) qName attributes:(NSDictionary *) attributeDict {
    //判断 elementName 字符串是否与 MacthingElement 相同
    if ([elementName isEqualToString:MacthingElement]) {
        //判断 _soapResults 是否为空
        if (!_soapResults) {
            _soapResults = [[NSMutableString alloc] init];
        }
        elementFound = YES;
    }
}

```

parser:didEndElement:namespaceURI:qualifiedName:方法实现结束解析这个元素名。程序代码如下：

```
-(void)parser:(NSXMLParser *)parser didEndElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName {
    //判断 elementName 字符串是否与 MacthingElement 相同
    if ([elementName isEqualToString:MacthingElement]) {
        self.soapBlock(_soapResults);
        //创建并设置警告视图对象
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"手机号码
        信息" message:[NSString stringWithFormat:@"%@", _soapResults]
        delegate:self cancelButtonTitle:@"确定" otherButtonTitles:nil];
        [alert show];
        elementFound = FALSE;
        //强制放弃解析
        [xmlParser abortParsing];
    }
}
```

(5) 创建一个基于 NSObject 类的 SCHttpClient 类。

(6) 打开 SCHttpClient.h 文件，编写代码，实现头文件、宏定义、类型定义、遵守协议、对象以及方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
#import "SoapNAL.h"
#define WebServicesURL @"http://webService.webxml.com.cn/WebServices/
MobileCodeWS.asmx"
typedef void(^SCHttpSuccessBlock)(NSString *soapResults);
@interface SCHttpClient : NSObject<NSURLConnectionDelegate>{
    NSMutableData *soapData;
}
-(void)postRequestWithPhoneNumber:(NSString *)number;
@end
```

(7) 打开 SCHttpClient.m 文件，编写代码，实现在客户端向 Web 服务发送参数请求的功能。使用的方法如表 12-2 所示。

表 12-2 SCHttpClient.m 文件中方法总结

方 法	功 能
postRequestWithPhoneNumber:	创建电话号码的请求
connection:didReceiveResponse:	在 Web 服务开始传递数据时调用
connection:didReceiveData:	在 Web 服务不断地传递数据时调用

其中，postRequestWithPhoneNumber:方法实现创建电话号码的请求。程序代码如下：

```
-(void)postRequestWithPhoneNumber:(NSString *)number{
    //创建 SOAP 信息
    NSString *soapMsg= [NSString stringWithFormat:
        @"<?xml version=\"1.0\" encoding=\"utf-8\"?>\"
        "<soap12:Envelope \"
        \"xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" \"
        \"xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" \"
        \"xmlns:soap12=\"http://www.w3.org/2003/05/soap-envelope\">\"
        "<soap12:Body>\"
        "<getMobileCodeInfo xmlns=\"http://WebXml.com.cn/\">\"
        "<mobileCode>%@</mobileCode>\"
```

```

        "<userID>%@", userID, "</userID>"
        "</getMobileCodeInfo>"
        "</soap12:Body>"
        "</soap12:Envelope>", number, @"");
NSLog(@"SOAPMsg==%@", soapMsg);
//创建 URL 请求
NSMutableURLRequest *request=[NSMutableURLRequest requestWithURL:
[NSURL URLWithString:WebServiceURL]];
NSString *msgLength=[NSString stringWithFormat:@"%d", [number length]];
//添加请求的详细信息, 与请求报文前半部分的各字段对应
[request addValue:@"application/soap+xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];
[request addValue:msgLength forHTTPHeaderField:@"Content-Length"];
[request setHTTPMethod:@"POST"]; //请求
[request setHTTPBody:[soapMsg dataUsingEncoding:4]]; //数据
NSURLConnection *connection=[[NSURLConnection alloc] initWithRequest:
request delegate:self];
if (connection) {
    soapData=[[NSMutableData alloc] init];
}
}

```

connection:didReceiveData:方法在 Web 服务不断地传递数据时调用,实现 XML 的打印。程序代码如下:

```

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data{
    [soapData appendData:data]; //添加数据
    NSString *theXML = [[NSString alloc] initWithBytes:[soapData mutableBytes] length:[soapData length] encoding:4];
    //打印出得到的 XML
    NSLog(@"得到的XML=%@", theXML);
    [[SOAPNal sharedInstance] parserSoapXML:soapData withParserBlock:^(NSString *parserXML) {
        NSLog(@"parserXML==%@", parserXML); //输出
    }];
}

```

(8) 打开 ViewController.h 文件,编写代码,实现头文件、插座变量以及动作的声明。程序代码如下:

```

#import <UIKit/UIKit.h>
#import "SCHttpClient.h"
@interface ViewController : UIViewController
{
    IBOutlet UITextField *phoneNumber;
}
- (IBAction)Query:(id) sender;
@end

```

(9) 打开 Main.storyboard 文件,对 View Controller 视图控制器的设计界面进行设计,效果如图 12.2 所示。

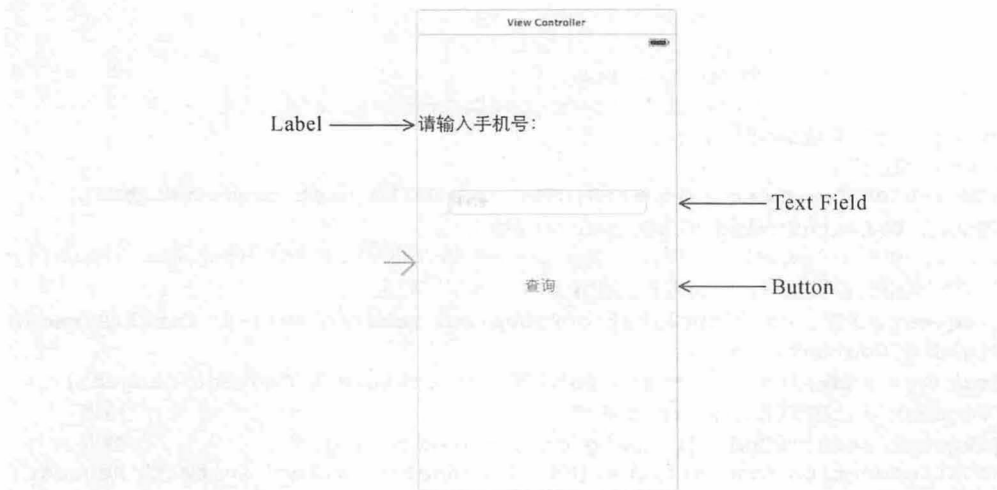


图 12.2 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 12-3 所示。

表 12-3 视图、控件设置

视图、控件	属 性 设 置	其 他
Label	Text: 请输入手机号: Font: System 21.0	
Text Field	Placeholder: 手机号 Keyboard: Number Pad	与插座变量 phoneNumber 关联
Button	Title: 查询 Font: System 19.0	与动作 Query:关联

(10) 打开 ViewController.m 文件，编写代码，实现电话号码的查询。程序代码如下：

```
- (IBAction)Query:(id)sender {
    SCHttpClient *client=[SCHttpClient new];           //创建 SCHttpClient 对象
    [client postRequestWithPhoneNumber:phoneNumber.text];
}
```

【代码解析】

本实例关键功能是 XML 解析和客户端向 Web 服务发送参数请求。下面依次讲解这两个知识点。

1. XML解析

XML 解析数据可以通过 SDK 自带的 NSXMLParser 以及 NSXMLParserDelegate 类实现。本实例也是使用这两个类实现了对从 Web 服务中获取的 XML 数据进行解析。本实例的解析思路是首先创建并设置一个 NSXMLParser 解析器。代码如下：

```
xmlParser=[[NSXMLParser alloc] initWithData:soapData];
xmlParser.delegate=self;
[xmlParser parse];
```

然后，开始解析一个元素名，就会调用 parser:didStartElement:namespaceURI:qualified

Name:attributes:方法。判断元素的名称是否和指定的元素名称匹配。代码如下:

```
if ([elementName isEqualToString:MacatchingElement]) {
    if (!_soapResults) {
        _soapResults = [[NSMutableString alloc] init];
    }
    elementFound = YES;
}
```

如果找到内容,就会调用 parser:foundCharacters:方法,将找到的内容追加到一个字符串中。程序代码如下:

```
if (elementFound) {
    [_soapResults appendString: string];
}
```

找到某一元素结尾处,就会调用 parser:didEndElement:namespaceURI:qualifiedName:方法,将解析的内容显示在警告视图中。代码如下:

```
if ([elementName isEqualToString:MacatchingElement]) {
    self.soapBlock(_soapResults);
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"手机号码信息"
        message:[NSString stringWithFormat:@"%s", _soapResults] delegate:self
        cancelButtonTitle:@"确定" otherButtonTitles:nil];
    [alert show];
    elementFound = FALSE;
    [xmlParser abortParsing];
}
```

2. 客户端向Web服务发送参数请求

在本实例中向 Web 服务发送参数请求,首先需要创建 SOAP 请求数据包。代码如下:

```
NSString *soapMsg= [NSString stringWithFormat:
    @"<?xml version=\"1.0\" encoding=\"utf-8\"?>"
    "<soap12:Envelope "
    "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" "
    "xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" "
    "xmlns:soap12=\"http://www.w3.org/2003/05/soap-envelope\">"
    "<soap12:Body>"
    "<getMobileCodeInfo xmlns=\"http://WebXml.com.cn/\">"
    "<mobileCode>%@", number, "</mobileCode>"
    "<userID>%@", number, "</userID>"
    "</getMobileCodeInfo>"
    "</soap12:Body>"
    "</soap12:Envelope>", number, @""];
```

其次使用 NSMutableURLRequest 类的 requestWithURL:URLWithString:方法创建一个 URL 请求。代码如下:

```
NSMutableURLRequest *request=[NSMutableURLRequest requestWithURL:
    [NSURL URLWithString:WebServiceURL]];
NSString *msgLength=[NSString stringWithFormat:@"%d",[number length]];
```

然后添加请求的详细信息。代码如下:

```
[request addValue:@"application/soap+xml; charset=utf-8" forHTTPHeaderField:@"Content-Type"];
[request addValue:msgLength forHTTPHeaderField:@"Content-Length"];
```



```
[request setHTTPMethod:@"POST"];
[request setHTTPBody:[soapMsg dataUsingEncoding:4]];
```

最后使用 `NSURLConnection` 的 `initWithRequest:delegate:` 方法实现对 `NSURLConnection` 对象的实例化，它是与 Web 服务建立连接的。代码如下：

```
NSURLConnection *connection=[[NSURLConnection alloc] initWithRequest:
request delegate:self];
```

实例 119 在 Safari 中打开 URL

【实例描述】

本实例实现的功能是在 Safari 应用程序中打开指定的 URL。当用户单击 Show Activities 按钮后，就会弹出分享列表，选择其中的 Open in Safari 动作就会在 Safari 中打开指定的 URL。运行效果如图 12.3 所示。



图 12.3 运行效果

【实现过程】

- (1) 创建一个项目，命名为“在 Safari 中打开 URL”。
- (2) 添加图像 1.png 到创建项目的 Supporting Files 文件夹中。
- (3) 创建一个基于 `UIActivity` 类的 `SafariActivity` 类。
- (4) 打开 `SafariActivity.h` 文件，编写代码，实现属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface SafariActivity : UIActivity
@property (nonatomic, strong) NSURL *url;
@end
```

- (5) 打开 `SafariActivity.m` 文件，编写代码，实现自定义的动作。

```
//获取类型
- (NSString *)activityType
{
    return NSStringFromClass([self class]);
}
```

```

}
//获取动作的图像
- (UIImage *)activityImage
{
    return [UIImage imageNamed:@"1.png"];
}
//获取动作的标题
- (NSString *)activityTitle
{
    return NSLocalizedStringFromTable(@"Open in Safari", NSStringFromClass(
        [self class]), nil);
}
//返回一个布尔值, 指示分享列表是否可以作用于指定的数据项
- (BOOL)canPerformWithActivityItems:(NSArray *)activityItems
{
    //遍历
    for (id activityItem in activityItems) {
        if ([activityItem isKindOfClass:[NSURL class]] && [[UIApplication
            sharedApplication] canOpenURL:activityItem]) {
            return YES;
        }
    }
    return NO;
}
//为动作指定数据服务
- (void)prepareWithActivityItems:(NSArray *)activityItems
{
    //遍历
    for (id activityItem in activityItems) {
        if ([activityItem isKindOfClass:[NSURL class]] && [[UIApplication
            sharedApplication] canOpenURL:activityItem]) {
            self.url = activityItem;           //设置 url
        }
    }
}
//执行服务
- (void)performActivity
{
    bool completed = NO;
    //判断 self.url 的值
    if (self.url) {
        completed = [[UIApplication sharedApplication] openURL:self.url];
    }
    [self activityDidFinish:completed];
}

```

(6) 打开 ViewController.m 文件, 编写代码, 实现按钮的创建以及显示分享列表等功能。程序代码如下:

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    //创建并设置按钮对象
    UIButton *showActivitiesButton = [UIButton buttonWithType:UIButtonType
        RoundedRect];
    [showActivitiesButton setTitle:@"Show Activities" forState:UIControlStateNormal];
    [showActivitiesButton addTarget:self action:@selector(showActivities:)

```

```

forControlEvents:UIControlEventTouchUpInside]; //添加动作
CGSize screenSize = [UIScreen mainScreen].bounds.size;
//设置框架
[showActivitiesButton setFrame:CGRectMake((screenSize.width/2)-60,
(screenSize.height/2)-30, 120, 60)];
[self.view addSubview:showActivitiesButton];
}
- (void)showActivities:(id)sender
{
    NSURL *url = [NSURL URLWithString:@"http://www.baidu.com"];
    //创建并显示分享列表
    SafariActivity *safariActivity = [[SafariActivity alloc] init];
    UIActivityViewController *activityVC = [[UIActivityViewController
alloc] initWithActivityItems:@[url] applicationActivities:@[safariActivity]];
[self presentViewController:activityVC animated:YES completion:nil];
//显示
}

```

【代码解析】

本实例关键功能是分享列表的自定义动作添加，以及在 Safari 中打开指定的 URL。下面依次讲解这两个知识点。

1. 分享列表的自定义动作添加

分享列表的自定义动作添加，首先需要创建一个基于 UIActivity 类的类，用于创建一个动作。创建之后，需要使用 UIActivityViewController 类将自定义的动作添加到分享列表中，这里需要使用 UIActivityViewController 的 initWithActivityItems: applicationActivities: 方法实现。它的功能是实例化一个 UIActivityViewController 对象，其语法形式如下：

```

- (id)initWithActivityItems:(NSArray *)activityItems applicationActivities:
(NSArray *)applicationActivities;

```

其中，(NSArray *)activityItems 表示要分享的东西；(NSArray *)applicationActivities 是数组对象，表示定制动作服务的支持。在本实例中就使用了 initWithActivityItems: applicationActivities: 方法将自定义的动作添加到分享列表中。代码如下：

```

UIActivityViewController *activityVC = [[UIActivityViewController alloc]
initWithActivityItems:@[url] applicationActivities:@[safariActivity]];

```

其中，@[url] 表示要分享的东西；@[safariActivity] 表示定制动作服务的支持。

2. 在 Safari 中打开指定的 URL

在本实例中当用户单击 Open in Safari 动作就会在 Safari 中打开指定的 URL。它的实现需要使用 UIApplication 的 openURL: 方法。它的功能是在指定的 URL 中打开资源。其语法形式如下：

```

- (BOOL)openURL:(NSURL *)url;

```

其中，url 表示指定的 URL。在本实例中的代码如下：

```

completed = [[UIApplication sharedApplication] openURL:self.url];

```

其中，self.url 表示指定的 URL。

实例 120 后台下载测试

【实例描述】

现在很多的应用程序都有后台运行的功能。当你长时间没有使用此应用，或者在某应用中进行下载时，都会在中通知中心中出现提示。本实例就为读者实现在后台下载的应用软件进行提示的功能。当用户单击“下载”按钮，进行应用软件下载后，退出此应用程序；当下载完成后，就会在中通知中心出现此应用程序下载完成的通知。运行效果如图 12.4 所示。



图 12.4 运行效果

【实现过程】

- (1) 创建一个项目，命名为“后台下载测试”。
- (2) 添加 AVFoundation.framework 到创建的项目中。
- (3) 打开 AppDelegate.h 文件，编写代码，实现属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface AppDelegate : UIResponder <UIApplicationDelegate>
@property (strong, nonatomic) UIWindow *window;
@property (copy) void (^backgroundSessionCompletionHandler)();
@end
```

- (4) 打开 AppDelegate.m 文件，编写代码，实现在为应用程序添加本地通知的功能。程序代码如下：

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Override point for customization after application launch.
    return YES;
}
//在切换到后台后调用，添加本地通知
- (void)application:(UIApplication *)application handleEventsForBackgroundURLSession:(NSString *)identifier
completionHandler:(void (^)(void))completionHandler {
    self.backgroundSessionCompletionHandler = completionHandler;
}
```

```

//添加本地通知
[self presentNotification];
}
//本地通知
-(void)presentNotification{
    UINotification* localNotification = [[UINotification alloc]
    init];
    localNotification.alertBody = @"下载完成!";           //设置提示主题
    localNotification.alertAction = @"后台传输下载已完成!"; //设置提示内容
    //提示音
    localNotification.soundName = UINotificationDefaultSoundName;
    //icon 提示加1
    localNotification.applicationIconBadgeNumber = [[UIApplication
    sharedApplication] applicationIconBadgeNumber] + 1;
    [[UIApplication sharedApplication] presentLocalNotificationNow:
    localNotification];
}

```

(5) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、插座变量、属性以及动作的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
//头文件
#import <AVFoundation/AVFoundation.h>
#import "AppDelegate.h"
@interface ViewController : UIViewController<NSURLSessionDelegate,
NSURLSessionTaskDelegate,
NSURLSessionDownloadDelegate,UIDocumentInteractionControllerDelegate>{
    IBOutlet UIProgressView *progressView;
}
//属性
@property (nonatomic)NSURLSession *session;
@property (nonatomic)NSURLSessionDownloadTask *downloadTask;
@property (strong,nonatomic)UIDocumentInteractionController *document
InteractionController;
- (IBAction)StartDownLoad:(id)sender;           //单击按钮实现下载的动作
@end

```

(6) 打开 Main.storyboard 文件，对 View Controller 视图控制器的界面进行设计，效果如图 12.5 所示。

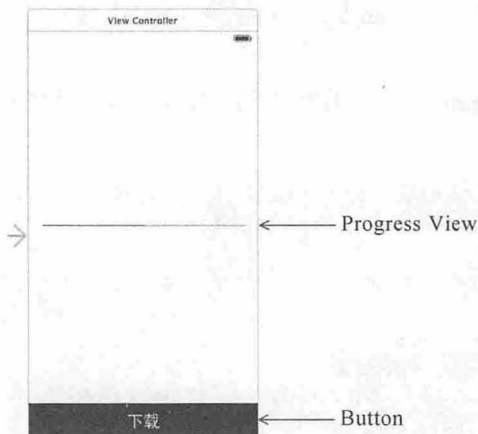


图 12.5 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 12-4 所示。

表 12-4 视图、控件设置

视图、控件	属 性 设 置	其 他
Progress View		与插座变量 progressView 关联
Button	Title: 下载 Font: System 23.0 Text Color: 白色 Background: 黑色	与动作 StartDownLoad:关联

(7) 打开 ViewController.m 文件, 编写代码, 实现后台下载测试的功能。使用的方法如表 12-5 所示。

表 12-5 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用, 实现初始化
StartDownLoad:	单击按钮后, 实现下载
backgroundSession	获取 session
NSURLSession:downloadTask:didWriteData:totalBytesWritten:totalBytesExpectedToWrite:	跟踪下载数据并且根据进度刷新进度条
NSURLSession:downloadTask:didFinishDownloadingToURL:	下载完成后调用
NSURLSession:task:didCompleteWithError:	完成任务的数据传输后调用
NSURLSessionDidFinishEventsForBackgroundNSURLSession:	一个会话结束之后, 会在后台调用

这里需要讲解几个重要的方法(其他方法请读者参考源代码)。其中, StartDownLoad:方法实现在单击按钮后, 实现下载的功能。程序代码如下:

```

- (IBAction)StartDownLoad:(id)sender {
    if (self.downloadTask) {
        return;
    }
    NSURL *downloadURL = [NSURL URLWithString:DownloadURLString];
    //创建 NSURL 对象
    NSURLRequest *request = [NSURLRequest requestWithURL:downloadURL];
    self.downloadTask = [self.session downloadTaskWithRequest:request];
    //创建下载任务
    [self.downloadTask resume];
    progressView.hidden = NO;
    //显示进度条
}

```

backgroundSession 方法实现对 session 的获取。程序代码如下:

```

- (NSURLSession *)backgroundSession {
    static NSURLSession *session = nil;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        //实例化对象 configuration
        NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration backgroundSessionConfigurationWithIdentifier:@"com.zyprosoft.backgroundsession"];
        session = [NSURLSession sessionWithConfiguration:configuration delegate:self delegateQueue:nil];
        //实例化对象 session
    });
}

```

```

});
return session;
}

```

NSURLSession:downloadTask:didWriteData:totalBytesWritten:totalBytesExpectedToWrite:方法用来跟踪下载数据，并且根据进度刷新进度条的显示。

```

- (void)NSURLSession:(NSURLSession *)session downloadTask:(NSURLSessionDownloadTask *)downloadTask didWriteData:(int64_t)bytesWritten totalBytesWritten:(int64_t)totalBytesWritten totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite
{
    //判断
    if (downloadTask == self.downloadTask) {
        double progress = (double)totalBytesWritten / (double)totalBytesExpectedToWrite;
        NSLog(@"下载任务: %@ 进度: %lf", downloadTask, progress);
        //实际成绩下载任务以及进度
        dispatch_async(dispatch_get_main_queue(), ^{
            progressBar.progress = progress; //刷新进度条
        });
    }
}

```

NSURLSession:downloadTask:didFinishDownloadingToURL:方法在下载完成后调用。它包含了已经完成下载任务的会话任务、下载任务和一个指向临时下载文件的文件路径。程序代码如下：

```

- (void)NSURLSession:(NSURLSession *)session downloadTask:(NSURLSessionDownloadTask *)downloadTask didFinishDownloadingToURL:(NSURL *)location
{
    NSFileManager *fileManager = [NSFileManager defaultManager]; //创建文件管理器
    //实例化对象
    NSArray *URLs = [fileManager URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask];
    NSURL *documentsDirectory = [URLs objectAtIndex:0];
    NSURL *originalURL = [[downloadTask originalRequest] URL];
    NSURL *destinationURL = [documentsDirectory URLByAppendingPathComponent:[originalURL lastPathComponent]];
    NSError *errorCopy;
    [fileManager removeItemAtURL:destinationURL error:NULL]; //移除指定的条目
    BOOL success = [fileManager copyItemAtURL:location toURL:destinationURL error:&errorCopy];
    //判断是否成功
    if (success) {
        dispatch_async(dispatch_get_main_queue(), ^{
        });
    } else {
        NSLog(@"复制文件发生错误: %@", [errorCopy localizedDescription]);
    }
}

```

NSURLSession:task:didCompleteWithError:方法在完成任任务的数据传输后调用，对进度条的进度进行刷新。程序代码如下：

```

- (void)NSURLSession:(NSURLSession *)session task:(NSURLSessionTask *)task didCompleteWithError:(NSError *)error{
    //判断是否出现错误
}

```

```

if (error == nil) {
    NSLog(@"任务: %@ 成功完成", task); //输出
} else {
    NSLog(@"任务: %@ 发生错误: %@", task, [error localizedDescription]); //输出
}
double progress = (double)task.countOfBytesReceived / (double)task.
countOfBytesExpectedToReceive;
dispatch_async(dispatch_get_main_queue(), ^{
    progressBar.progress = progress; //设置进度
});
self.downloadTask = nil;
}

```

NSURLSessionDidFinishEventsForBackgroundURLSession:方法实现一个会话结束之后,在后台调用。程序代码如下:

```

- (void)NSURLSessionDidFinishEventsForBackgroundURLSession: (NSURLSession *)
session
{
    AppDelegate *appDelegate = (AppDelegate *)[[UIApplication shared
Application] delegate];
    if (appDelegate.backgroundSessionCompletionHandler) {
        void (^completionHandler)() = appDelegate.backgroundSession
CompletionHandler;
        appDelegate.backgroundSessionCompletionHandler = nil;
        completionHandler(); //设置后台会话完成时的处理
    }
    NSLog(@"所有任务已完成!");
}

```

【代码解析】

本实例关键功能是后台下载和在通知中心中显示。下面依次讲解这两个知识点。

1. 后台下载

后台下载也是数据传输的一种,即需要 NSURLSession 类,它的实现首先需要创建一个 NSURLSessionConfiguration 对象,它用于为 NSURLSession 设置工作模式。其中它的工作模式有 3 种,如表 12-6 所示。

表 12-6 工作模式

模 式	功 能
一般模式	工作模式类似于原来的 NSURLConnection, 可以使用缓存的 Cache、Cookie、鉴权
及时模式	不使用缓存的 Cache、Cookie、鉴权
后台模式	在后台完成上传下载

其次就是创建一个 NSURLSession 对象,此对象用于数据传输。在数据传输时,需要实现某一种任务。在本实例中使用的是 NSURLSessionDownloadTask 下载任务(有两种:一种是 NSURLSessionDataTask 数据任务,另一种是 NSURLSessionUploadTask 上传任务)。然后为使用 NSURLSession 的 downloadTaskWithRequest:方法创建一个下载任务,其语法形式如下:

```

- (NSURLSessionDownloadTask *)downloadTaskWithRequest: (NSURLRequest *)request;

```

其中，(NSURLRequest *)request 表示一个 NSURLRequest 对象，它提供了 URL。在本实例中就使用了此方法创建了一个下载任务。代码如下：

```
self.downloadTask = [self.session downloadTaskWithRequest:request];
```

最后就是使用 resume 方法恢复下载任务（开启下载任务），其语法形式如下：

```
-(void)resume;
```

在本实例中就是使用了 resume 方法开始了下载任务。代码如下：

```
[self.downloadTask resume];
```

2. 在通知中心显示

在应用软件下载完成后，用户的应用将会重启，并传输内容过来，这时就是调用 application: handleEventsForBackgroundURLSession: 方法。它的功能就是处理后台会话的事件，在本实例中就使用了 application: handleEventsForBackgroundURLSession: 方法调用了本地通知 presentNotification 方法。在此方法中实现了通知在通知中心显示的功能。首先，创建并设置一个 UILocalNotification 通知对象，然后使用 presentLocalNotificationNow: 方法显示本地通知。代码如下：

```
-(void)presentNotification{
    UILocalNotification* localNotification = [[UILocalNotification alloc] init];
    .....
    [[UIApplication sharedApplication] presentLocalNotificationNow:
    localNotification];
}
```

实例 121 图像下载队列控制器

【实例描述】

本实例主要实现一个图像下载队列控制器。当用户运行程序，就可以看到很多的图像下载地址，并且有图像开始下载；如果下载成功，图像就会显示在 Image View 图像视图中。运行效果如图 12.6 所示。



图 12.6 运行效果

【实现过程】

- (1) 创建一个项目，命名为“图像下载队列控制器”。
- (2) 添加第三方库 AFNetworking 到创建的项目中。
- (3) 创建一个基于 NSObject 类的 ImgOperator 类。
- (4) 打开 ImgOperator.h 文件，编写代码，实现头文件、协议、属性以及方法的声明。

程序代码如下：

```
#import <Foundation/Foundation.h>
//方法
#import "AFHTTPRequestOperation.h"
@class ImgOperator;
//协议
@protocol ImgOperatorDelegate <NSObject>
- (void)remoteImgOper:(ImgOperator *)oper getImgSucc:(NSData *)dataImg
fromURL:(NSString *)strURL;
- (void)remoteImgOper:(ImgOperator *)oper getImgFailedFromURL:(NSString
*)strURL;
@end
@protocol DownloadImgProgressDelegate <NSObject>
- (void)setProgress:(float)newProgress; //设置进度
@end
@interface ImgOperator : NSObject
//属性
@property (unsafe_unretained) id<ImgOperatorDelegate>delegate;
@property (nonatomic, readonly) AFHTTPRequestOperation *m_objAFOper;
@property (nonatomic, readonly) id <DownloadImgProgressDelegate>
downloadProgressDelegate;
// 方法
- (BOOL)getRemoteImgFromURL:(NSString *)strSrcURL; //从网络获取图像
- (BOOL)getRemoteImgFromURL:(NSString *)strSrcURL progressDelegate:
(id)progress;
- (void)cancelRequest; //取消下载
- (void)setProgressDelegate:(id)progress;
- (id)getProgressDelegate;
@end
```

(5) 打开 ImgOperator.m 文件，编写代码，实现图像下载操作的功能。使用的方法如表 12-7 所示。

表 12-7 ImgOperator.m 文件中方法总结

方 法	功 能
init	初始化
getRemoteImgFromURL:	从网络获取图像
getRemoteImgFromURL: progressDelegate:	从网络获取图像，带进度条委托
cancelRequest	取消下载
setProgressDelegate:	设置进度条委托
getProgressDelegate	获取进度条委托

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，getRemoteImgFromURL: progressDelegate:方法实现从网络获取图像的功能，并带有进度条的委托。程序代码如下：


```

- (BOOL)getRemoteImgFromURL:(NSString *)strSrcURL progressDelegate:(id)
progress
{
    BOOL bRet = NO;
    [self cancelRequest];
    if (strSrcURL && (strSrcURL.length > 0))
    {
        bRet = YES;
        [self cancelRequest];
        downloadProgressDelegate = progress;           //设置委托
        __block NSString *blockStrURL = [strSrcURL copy];
        __weak typeof(self) blockSelf = self;
        //创建并设置 AFHTTPRequestOperation 对象
        _objAFOper = [[AFHTTPRequestOperation alloc] initWithRequest:
        [NSURLRequest requestWithURL:[NSURL URLWithString:strSrcURL]]];
        _objAFOper.responseSerializer = [AFHTTPResponseSerializer serializer];
        //设置完成调用的块
        [_objAFOper setCompletionBlockWithSuccess:^(AFHTTPRequestOperation
        *operation, id responseObject){
            @try {
                NSData *data = (NSData *)responseObject;    //实例化数据对象
                if (data)
                {
                    if (blockSelf.delegate && [blockSelf.delegate responds
                    ToSelector:@selector(remoteImgOper:getImgSucc:fromURL:)])
                    {
                        //通知获取成功
                        [blockSelf.delegate remoteImgOper:blockSelf getImgSucc:
                        data fromURL:blockStrURL];
                    }else{}
                }
                else
                {
                    if (blockSelf.delegate && [blockSelf.delegate responds
                    ToSelector:@selector(remoteImgOper:getImgFailedFromURL:)])
                    {
                        //通知获取失败
                        [blockSelf.delegate remoteImgOper:blockSelf getImg
                        FailedFromURL:blockStrURL];
                    }else{}
                }
            }
            @catch (NSEException *exception) {
                if (blockSelf.delegate && [blockSelf.delegate respondsTo
                Selector:@selector(remoteImgOper:getImgFailedFromURL:)])
                {
                    //通知获取失败
                    [blockSelf.delegate remoteImgOper:blockSelf getImg
                    FailedFromURL:blockStrURL];
                }else{}
            }
            @finally {}
        }
        failure:^(AFHTTPRequestOperation *operation, NSError *error)
        {
            if (blockSelf.delegate && [blockSelf.delegate respondsTo
            Selector:@selector(remoteImgOper:getImgFailedFromURL:)])
            {
                //通知获取失败

```

```

        [blockSelf.delegate remoteImgOper:blockSelf getImgFailed
        FromURL:blockStrURL];
    }else{}
    }];
    //设置下载进度的块
    [_objAFOper setDownloadProgressBlock:^(NSUInteger bytesRead, long
    long totalBytesRead, long long totalBytesExpectedToRead){
        if (blockSelf.downloadProgressDelegate)
        {
            CGFloat f = (CGFloat)totalBytesRead / totalBytesExpectedToRead;
            [blockSelf.downloadProgressDelegate setProgress:f]; //设置进度
        }else{}
    }];
    [_objAFOper start];
}
else
{
    bRet = NO;
}
return bRet;
}

```

(6) 创建一个基于 NSObject 类的 ImgListOperator 类。

(7) 打开 ImgListOperator.h 文件, 编写代码, 实现头文件、宏定义、遵守协议、属性以及方法的声明。程序代码如下:

```

#import <Foundation/Foundation.h>
#import "ImgOperator.h"
//宏定义
#define STR_ListElementURL                @"ImgURL"
#define STR_ListElementRequest            @"HTTPRequest"
#define INT_DefaultListSize                10
@interface ImgListOperator : NSObject<ImgOperatorDelegate>
//属性
@property (nonatomic, readonly) NSString *m_strSuccNotificationName;
@property (nonatomic, readonly) NSString *m_strFailedNotificationName;
.....
@property (nonatomic, readonly) NSInteger m_iListSize;
//方法
- (void)resetListSize:(NSInteger)iSize;                //设置列表最大长度
- (void)getRemoteImgByURL:(NSString *)strURL;            //从网络下载图片
- (void)getRemoteImgByURL:(NSString *)strURL withProgress:(id)progress;
- (void)removeProgressDelegate:(id)progress;
@end

```

(8) 打开 ImgListOperator.m 文件, 编写代码, 实现图像列表下载操作的功能。使用的方法如表 12-8 所示。

表 12-8 ImgListOperator.m 文件中方法总结

方 法	功 能
init	初始化
resetListSize:	设置列表最大长度
getRemoteImgByURL:	从网络下载图片
getRemoteImgByURL:withProgress:	从网络下载图片, 带进度条委托
removeRemoteOperFromListByURL:	移除网络下载的请求的图像
removeAllProgressDelegate	移除所有的进度条委托

续表

方 法	功 能
removeProgressDelegate:	移除正在使用的进度条委托
remoteImgOper:getImgSucc:	获取通知成功
remoteImgOper:getImgFailedFromURL:	获取通知失败

其中，init 方法实现对图像下载列表初始化的功能。程序代码如下：

```

- (id)init
{
    self = [super init];
    if (self)
    {
        static int s_iObjbTag = 0;
        s_iObjbTag++;
        _queueRemoteImgOper = dispatch_queue_create("com.company.app.
        remoteImgOperList", NULL);
        _arrRemoteImgOper = [[NSMutableArray alloc] init]; //创建可变数组
        //创建字符串对象
        _strSuccNotificationName = [NSString stringWithFormat:@"Remote
        ImgOperListSucc%d", s_iObjbTag];
        _strFailedNotificationName = [NSString stringWithFormat:@"Remote
        ImgOperListFailed%d", s_iObjbTag];
        _iListSize = INT_DefaultListSize;
    }else{}
    return self;
}

```

getRemoteImgByURL: withProgress:方法是在从网络下载图片，并带进度条委托。程序代码如下：

```

- (void)getRemoteImgByURL: (NSString *)strURL withProgress: (id)progress
{
    //判断 strURL 是否不为空并且 strURL 的长度大于 0
    if (strURL && strURL.length > 0)
    {
        __block NSString *strBlockURL = [strURL copy];
        __weak id progressBlock = progress;
        //将 block 发送到指定的线程去执行
        dispatch_sync(_queueRemoteImgOper, ^{
            BOOL bIsRequesting = NO;
            //遍历
            for (NSDictionary *dicItem in _arrRemoteImgOper)
            {
                NSString *strElementURL = [dicItem objectForKey:STR_
                ListElementURL];
                //判断 strElementURL 是否不为空并且 strElementURL 和 strBlockURL 相同
                if (strElementURL && [strElementURL isEqualToString:strBlockURL])
                {
                    ImgOperator *objImgOper = [dicItem objectForKey:
                    STR_ListElementRequest];
                    //判断 progressBlock 是否不为空
                    if (progressBlock)
                    {
                        [objImgOper setProgressDelegate:progressBlock];
                        //设置委托
                    }else{}
                }
            }
        });
    }
}

```

```

        bIsRequesting = YES;
        break;
    }else{}
}
//判断 bIsRequesting 是否为 NO
if (!bIsRequesting)
{
    ImgOperator *objImgOper = [[ImgOperator alloc] init];
    [objImgOper setDelegate:self]; //设置委托
    NSMutableDictionary *dicElement = [[NSMutableDictionary alloc] init];
    [dicElement setObject:[strBlockURL copy] forKey:STR_ListElementURL];
    [dicElement setObject:objImgOper forKey:STR_ListElementRequest];
    [_arrRemoteImgOper addObject:dicElement]; //添加对象
    [objImgOper getRemoteImgFromURL:strBlockURL progressDelegate:
    progressBlock];
    //判断是否 _arrRemoteImgOper 不为空, 并且 _arrRemoteImgOper 中元素
    个数大于 _iListSize
    if (_arrRemoteImgOper && _arrRemoteImgOper.count > _iListSize)
    {
        //列表满, 取消第一个的下载并退出
        NSDictionary *dicFirst = [_arrRemoteImgOper objectAtIndex:0];
        //判断 dicFirst 是否不为空
        if (dicFirst)
        {
            ImgOperator *objOper = [dicFirst objectForKey:STR_
            ListElementRequest];
            //判断 objOper 是否不为空
            if (objOper)
            {
                [objOper cancelRequest]; //取消下载
                objOper = nil;
            }else{}
        }else{}
        [_arrRemoteImgOper removeObjectAtIndex:0]; //移除指定的对象
    }else{}
}
});
}else{}
}

```

removeRemoteOperFromListByURL:方法实现移除网络下载的请求的图像。程序代码如下:

```

- (void)removeRemoteOperFromListByURL:(NSString *)strURL
{
    dispatch_sync(_queueRemoteImgOper, ^{
        //遍历
        for (NSDictionary *dicItem in _arrRemoteImgOper)
        {
            NSString *strElementURL = [dicItem objectForKey:STR_ListElementURL];
            //判断 strElementURL 是否不为空, 并且 strElementURL 与 strURL 相同
            if (strElementURL && [strElementURL isEqualToString:strURL])
            {
                ImgOperator *objImgOper = [dicItem objectForKey:
                STR_ListElementRequest];
                [objImgOper setProgressDelegate:nil]; //设置委托
                [objImgOper cancelRequest]; //取消下载
                [_arrRemoteImgOper removeObject:dicItem]; //移除对象
            }
        }
    });
}

```



```

        break;
    }else{}
}
});
}

```

(9) 创建一个基于 UITableViewCell 类的 ImageCell 类。

(10) 打开 ImageCell.h 文件，编写代码，实现头文件、插座变量、属性以及方法的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
@class ImgListOperator;
@interface ImageCell : UITableViewCell
{
    //插座变量
    IBOutlet UIImageView *m_imgView;
    IBOutlet UILabel *m_labelMsg;
}
//属性
@property (nonatomic, readonly) ImgListOperator *m_objRemoteImgListOper;
@property (nonatomic, readonly, copy) NSString *m_strURL;
//方法
- (void)setRemoteImgOper:(ImgListOperator *)objOper;//设置远程操作图像的功能
- (void)showImgByURL:(NSString *)strURL;
@end

```

(11) 打开 ImageCell.m 文件，编写代码，实现对表视图单元格的设置。使用的方法如表 12-9 所示。

表 12-9 ImageCell.m文件中方法总结

方 法	功 能
setSelected:animated:	设置选择状态以及动画
prepareForReuse	准备一个即将出列的单元
setRemoteImgOper:	设置远程操作图像的功能
showImgByURL:	显示图像
remoteImgSucc:	响应下载成功的通知，并显示图片
remoteImgFailed:	响应下载失败的通知，并在标签栏中显示下载失败的信息

setRemoteImgOper:方法实现对远程操作图像的设置功能。程序代码如下：

```

- (void)setRemoteImgOper:(ImgListOperator *)objOper
{
    //判断_objRemoteImgListOper 是否不等于 objOper
    if (_objRemoteImgListOper != objOper)
    {
        判断_objRemoteImgListOper 是否不为空
        if (_objRemoteImgListOper)
        {
            //注册通知
            [[NSNotificationCenter defaultCenter] removeObserver:self name:
            _objRemoteImgListOper.m_strSuccNotificationName object:nil];
            //注册通知
            [[NSNotificationCenter defaultCenter] removeObserver:self name:

```



```

        _objRemoteImgListOper.m_strFailedNotificationName object:nil];
    }else{}
    _objRemoteImgListOper = objOper;
    //判断_objRemoteImgListOper 是否不为空
    if (_objRemoteImgListOper)
    {
        //注册通知
        [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(remoteImgSucc:) name:_objRemoteImgListOper.m_strSucc
        NotificationName object:nil];
        //注册通知
        [[NSNotificationCenter defaultCenter] addObserver:self selector:
        @selector(remoteImgFailed:) name:_objRemoteImgListOper.m_strFailed
        NotificationName object:nil];
    }else{}
    }else{}
}

```

showImgByURL:方法实现对图像的显示。程序代码如下:

```

- (void) showImgByURL:(NSString *)strURL
{
    _strURL = strURL ? strURL : @"";
    m_labelMsg.text = [NSString stringWithFormat:@"开始下载...%@", _strURL];
    //设置文本内容
    __block NSString *blockStrURL = [strURL copy];
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_
    DEFAULT, 0), ^{
        dispatch_sync(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_
        DEFAULT, 0), ^{
            //判断 blockStrURL 的长度是否大于 1
            if (blockStrURL.length > 1)
            {
                // 从网络下载
                dispatch_async(dispatch_get_main_queue(), ^{
                    //判断_objRemoteImgListOper 是否不为空
                    if (_objRemoteImgListOper)
                    {
                        [_objRemoteImgListOper getRemoteImgByURL:blockStrURL
                        withProgress:nil];
                    }else{}
                });
            }else{}
        });
    });
}

```

remoteImgSucc:方法在下载成功后调用, 将下载的图像显示出来。程序代码如下:

```

- (void) remoteImgSucc:(NSNotification *)noti
{
    if (noti && noti.userInfo && noti.userInfo.allKeys && (noti.userInfo.
    allKeys.count > 0))

```

```

{
    NSString *strURL;
    NSData *dataImg;
    strURL = [noti.userInfo.allKeys objectAtIndex:0];
    dataImg = [noti.userInfo objectForKey:strURL];
    //判断_strURL 是否不为空，并且_strURL 和 strURL 相同
    if (_strURL && [_strURL isEqualToString:strURL])
    {
        m_labelMsg.text = [NSString stringWithFormat:@"下载成功 %@",
            _strURL]; //设置文本内容
        m_imgView.image = [UIImage imageWithData:dataImg]; //设置图像
    }else{}

}
}

```

remoteImgFailed:方法在下载失败后调用，并在标签栏中显示下载失败的信息。程序代码如下：

```

- (void)remoteImgFailed:(NSNotification *)noti
{
    if (noti && noti.userInfo && noti.userInfo.allKeys && (noti.userInfo.allKeys.count > 0))
    {
        NSString *strURL;
        strURL = [noti.userInfo.allKeys objectAtIndex:0];
        //判断_strURL 是否不为空，并且_strURL 和 strURL 相同
        if (_strURL && [_strURL isEqualToString:strURL])
        {
            m_labelMsg.text = [NSString stringWithFormat:@"下载失败 %@",
                _strURL]; //设置文本内容
        }else{}

    }else{}
}

```

(12) 打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、插座变量以及属性的声明。程序代码如下：

```

#import <UIKit/UIKit.h>
#import "ImgListOperator.h"
#import "ImageCell.h"
@interface ViewController : UIViewController<UITableViewDataSource,
UITableViewDelegate>{
    //插座变量
    IBOutlet UITableView *m_tableView;
    IBOutlet UILabel *m_labelMsg;
}
//属性
@property (nonatomic, readonly) NSMutableArray *m_arrImgURLs;
@property (nonatomic, readonly) ImgListOperator *m_objImgListOper;
@end

```

(13) 打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 12.7 所示。

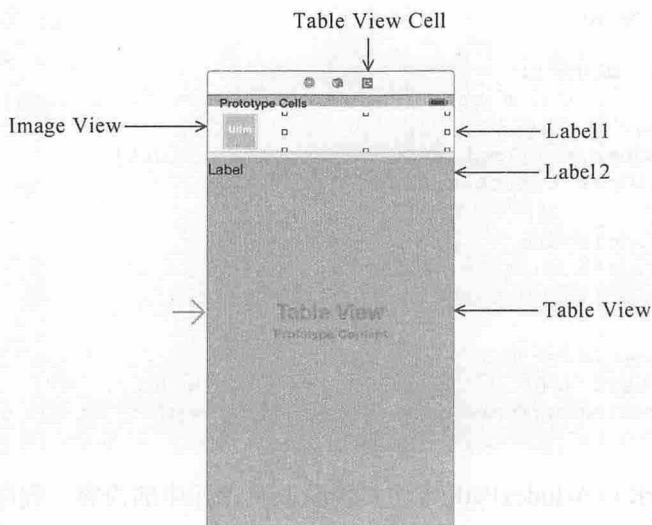


图 12.7 View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 12-10 所示。

表 12-10 视图、控件设置

视图、控件	属 性 设 置	其 他
Label1	Text: (空) Font: System 15.0 Lines: 2	与插座变量 m_labelMsg (Image Cell) 关联
Label2	Font: System 19.0 Lines: 2	与插座变量 m_labelMsg (View Controller) 关联
Image View		与插座变量 m_imgView 关联
Table View		与插座变量 m_tableView 关联
Table View Cell	Identifier: ImageCell	Class: ImageCell

(14) 打开 ViewController.m 文件，编写代码，图像对图像下载队列控制器的设置。使用的方法如表 12-11 所示。

表 12-11 ViewController.m文件中方法总结

方 法	功 能
viewDidLoad	视图加载后调用，实现初始化
tableView:heightForRowAtIndexPath:	获取行高
numberOfSectionsInTableView:	获取块数
tableView:numberOfRowsInSection:	获取行数
tableView:cellForRowAtIndexPath:	获取内容
loadDataSource	加载数据
dataSourceDidLoad	数据加载成功
dataSourceDidError	数据加载错误

其中，viewDidLoad 方法在视图加载后调用，实现初始化的功能。

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    // 创建一个队列对象，以便在当前页面内统一控制下载数量。
    _objImgListOper = [[ImgListOperator alloc] init];
    [_objImgListOper resetListSize:20];
    // 设置表视图
    m_tableView.delegate = self;
    m_tableView.dataSource = self;
    _arrImgURLs = [[NSMutableArray alloc] init];
    // 设置标签
    m_labelMsg.hidden = NO;
    m_labelMsg.text = @" 正在从 imgur.com 获取图片 URL ... ";
    [self performSelector:@selector(loadDataSource) withObject:nil afterDelay:0.3];
}

```

tableView: cellForRowAtIndexPath: 方法实现获取表单元中的内容。程序代码如下：

```

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"ImageCell";
    ImageCell *cell = (ImageCell *)[tableView dequeueReusableCellWithIdentifier:
    Identifier:CellIdentifier];
    NSString *strURL = [_arrImgURLs objectAtIndex:indexPath.row];
    [cell setRemoteImgOper:_objImgListOper];           // 设置远程操作图像
    [cell showImgByURL:strURL];                         // 显示图像
    return cell;
}

```

loadDataSource 方法实现对数据的加载。程序代码如下：

```

- (void)loadDataSource
{
    // 请求
    NSString *URLPath = [NSString stringWithFormat:@"http://imgur.com/
    gallery.json"];
    NSURL *URL = [NSURL URLWithString:URLPath];
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:URL];
    // 异步请求
    [NSURLConnection sendAsynchronousRequest:request queue:[NSOperation
    Queue mainQueue] completionHandler:^(NSURLResponse *response, NSData
    *data, NSError *error) {
        NSInteger responseCode = [(NSHTTPURLResponse *)response statusCode];
        // 判断 error 是否为空并且 responseCode 是否等于 200
        if (!error && responseCode == 200) {
            id res = [NSJSONSerialization JSONObjectWithData:data options:
            NSJSONReadingMutableContainers error:nil];           // 解析
            // 判断 res 是否不为空，并且 res 是否在 NSDictionary 中
            if (res && [res isKindOfClass:[NSDictionary class]]) {
                NSArray *arrItems = [res objectForKey:@"data"];
                // 判断 arrItems 是否不为空
                if (arrItems)
                {
                    // 遍历
                    for (NSDictionary *item in arrItems)
                    {

```

```

        [_arrImgURLs addObject:[NSString stringWithFormat:
@"http://i.imgur.com/%@%@", [item objectForKey:@"hash"],
[item objectForKey:@"ext"]]]; //添加对象
    }
    }else{}
    [self dataSourceDidLoad]; //调用数据加载成功的方式
} else {
    [self dataSourceDidError]; //调用数据加载错误的方法
}
} else {
    [self dataSourceDidError]; //调用数据加载错误的方法
}
}
}];
}

```

【代码解析】

本实例关键功能是图像的获取。下面就是这个知识点的详细讲解。

在本实例中使用第三方库 AFNetworking 实现网络编程。对于图像的获取是使用 AFURLConnectionOperation（是一个网络请求）的子类 AFHTTPRequestOperation 实现的。它针对 HTTP+HTTPS 协议做了封装。在使用的时候，首先需要进行对象的实例化。在本实例中，实例化代码如下：

```

_objAFOper = [[AFHTTPRequestOperation alloc] initWithRequest:[NSURLRequest
requestWithURL:[NSURL URLWithString:strSrcURL]]];

```

对于图像的获取实现，通过 AFHTTPRequestOperation 类添加的请求成功和失败的回调处理块实现的。它们分别对请求成功或者失败进行处理，在本实例中的代码如下：

```

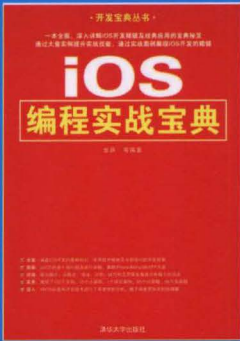
[_objAFOper setCompletionBlockWithSuccess:^(AFHTTPRequestOperation *operation,
id responseObject)
{
    .....
}
failure:^(AFHTTPRequestOperation *operation, NSError *error)
{
    .....
}];

```

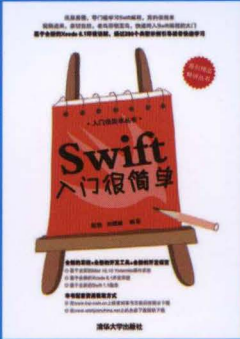

移动开发精品图书



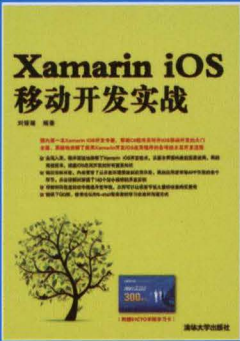
以全新的Xcode 6为开发环境讲解iOS 8应用开发
用117个实例全面展现iOS开发的14类界面模块



iOS百科全书，全面、新颖、详细、实用、深入
155个实例、35个案例、1个项目案例、91个习题



基于全新的Xcode 6.1环境和Mac 10.10 Yosemite系统
基于全新的Swift正式版，详解290个典型示例



国内第一本Xamarin iOS开发专著，涵盖182个开发实例
帮助C#程序员叩开iOS移动开发的大门

iOS 开发 范例实战宝典（进阶篇）

本书涵盖的精华内容

- ☑ 图形图像（39个实例）
- ☑ 图表（5个实例）
- ☑ 动画（16个实例）
- ☑ 网页视图（9个实例）
- ☑ 地图（13个实例）
- ☑ 音频和视频（9个实例）
- ☑ 内置的应用程序（4个实例）
- ☑ 触摸和手势（8个实例）
- ☑ 照片库与相机（10个实例）
- ☑ 传感器（4个实例）
- ☑ 网络（4个实例）

本书配套资源获取方式

本书涉及的源程序等资源需要读者自行下载。请到清华大学出版社的网站上搜索到本书页面，然后按照提示下载即可。读者也可到本书服务网站上的相关版块下载。具体见本书前言中的说明。

清华大学出版社数字出版网站

WQBook  书文局泉

www.wqbook.com

上架建议：计算机/移动开发

ISBN 978-7-302-39702-1



9 787302 397021 >

定价：99.00元